HP-41Z Module

Complex Number Module for the HP-41

User's Manual and Quick Reference Guide

Written and developed by: Ángel M. Martin

October 2011

This compilation, revision A.3.5.

The author wishes to thank the contributors to this project in various ways, as follows:

W. Doug Wilder, who wrote the code for the non-merged functions in program mode,
Håkan Thörngren for his assistance and advices on the Memory Buffer implementation,
Valentín Albillo, who wrote the original "ZPROOT" FOCAL program,
M. Luján García, who prepared the 41Z Keys overlay bitmap file.

Some graphics taken from http://www.clarku.edu/~djoyce/complex, copyright 1999 by David E. Joyce.
Some graphics taken from http//www.wikipedia.org

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow. See http://www.hp41.org/

Index.

41Z – Complex Number Module for the HP-41

1. Introduction.

Complex Number handling is perhaps one of the very few areas where the HP-41 didn't have a comprehensive set of native functions, written in machine mode and so taking advantage of the speed and programming enhancements derived from it. While both the Math Pack and the Advantage Rom provide FOCAL programs for complex number treatment, neither of them could be properly consider as a full function set to the effect of, for instance, the powerful Matrix handling functions contained in the Advantage Rom (in turn an evolution of those implemented in the CCD Module).

The 41Z module provides a significant number of functions that should address the vast majority of complex number problems, in a user-friendly context, and with full consistency. To that goal this manual should also contribute to get you familiar with their usage and applications, hopefully learning a couple of new things and having some fun during the process.

The implementation provided in this 8k-module is a second-generation code, building on the initial 41Z ROM released by the author in April 2005. Numerous improvements have been added to the initial function set, notably the addition of a *4-level complex stack, a POLAR mode,* and a fully featured *complex mode keyboard*. Memory management is facilitated by prompting functions that deal with complex arguments, like **ZSTO**, **ZSTO** Math, **ZRCL**, **Z<>,** and **ZVIEW** - all of them fully programmable as well.

2. Complex Stack, number entering and displaying.

A four-level complex stack is available to the user to perform all complex calculations. The complex stack levels are called ***U***, ***V***, ***W***, and ***Z*** – from top to bottom. Each level holds two real numbers, the imaginary and real parts of the corresponding complex number. Besides them, a "LastZ" complex register ***S*** temporarily stores the argument of the last executed function.

	41Z Complex Stack	
b11	*non-zero*	
b10	U	-
b9		-
b8	V	-
b7		-
b6	W	*T*
b5		*Z*
b4	Z	*Y*
b3		*X*
b2	(S)	-
b1		*L*
b0	*Header*	

The complex stack uses a dedicated buffer in main memory. It is created and maintained by the 41Z module and its operation should be transparent to the user. This buffer is independent from the real stack (X, Y, Z, and T registers) but it's important however to understand how they interact with each other. A complex number uses two real stack levels (like X and Y), but a single complex stack level (like ***Z*** or ***W***). The figure on the left shows the relationship between the complex and real stacks, which is automatically maintained upon function execution, as we'll see later on.

The real stack is used to enter the complex number values, real and imaginary parts. The input sequence varies depending on the method used *but all functions will expect the imaginary part in the Y register and the real part in the X register.* More about this later.

The contents of complex and real stack levels are *automatically synchronized* before and after each complex operation is performed. This may just involve real levels X,Y and complex level ***Z*** if

it's a monadic (or unary) operation requiring a single complex argument, or may also involve real levels Z,T and complex level ***W*** if it's a dual operation requiring two complex arguments.

Monadic functions will assume that the real numbers in X,Y are the most-updated values for the real and imaginary parts of the complex argument. They will overwrite the contents of complex level ***Z***. This allows quick editing and modification of the complex argument prior to executing the function.

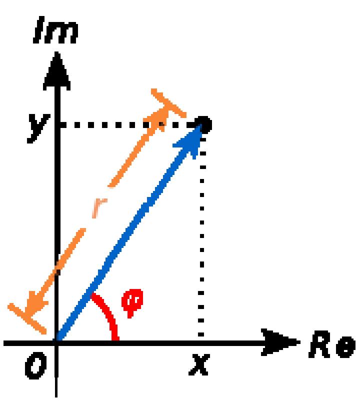
Dual functions will assume that the second argument is stored in ***W***, that is level 2 of the complex stack, and *will thus ignore the values contained in real stack registers Z,T*. Note that because the real stack overflows when trying to hold more than four different values, it is not a reliable way to input two complex numbers at once.

The design objective has been to employ as much as possible the same rules and conventions as for the real number stack, only for complex numbers instead. This has been accomplished in all aspects of data entering, *with the exception of automated complex stack lift*: with a few exceptions, entering two complex numbers into the complex stack requires pressing **ZENTER^** to separate them.

Once again: entering two complex numbers into the complex stack is accomplished by executing **ZENTER^** to separate the first and second complex number. Exceptions to this rule are the other complex-stack lifting functions, such as **GEUZ**, **ZRCL**, **ZRPL^**, **IMAGINE**, **^ZIMAG**, **^ZREAL**, **^IM/AG**, and the "***Complex Keypad***". Here the left-side symbol "^" (SHIFT-N) represents an input action.

2.1 Rectangular vs. Polar forms.

The HP-41 sorely lacks a polar vs. Rectangular mode. This limitation is also overcome on the 41Z module, with the functions **POLAR** and **RECT** to switch back and forth between these modes. It uses an internal flag in the complex buffer, not part of the 41 system flags. The operation is simplified in that *complex numbers are always stored in their rectangular (or Cartesian) form*, *z*=x+yi. So while all functions expect the argument(s) in rectangular form, yet *the results are shown in the appropriate format as defined by the POLAR or RECT mode*. (The notable exception is **ZPOL**, which always returns the value in Polar form). Note also that the POLAR mode is directly affected by the angular mode as well, as it occurs with real argument values.



Note: The POLAR display of the complex number requires an additional R-P conversion after the result is calculated in Cartesian form. The Polar form is temporarily stored in the Real stack registers T,Z – *which have no active role in the Complex Stack* and therefore can always be used as scratch. Once again, no changes are made to either X,Y registers or Complex stack level **Z**.

2.2 Data Entry Conventions

And how about complex number entering? Here the world divides in two camps, depending on whether the sequence is: "Re(*z*), ENTER^, Im(*z*)" – like on the HP-42S - , or its reverse: "Im(*z*), ENTER^, Re(*z*)" – like on the HP-32/33S and other FOCAL programs -. With the 41Z module you can do it either way, but it's important to remember that *regardless of how you introduce the numbers, all functions expect the imaginary part in the Y real-stack register and the real part in the X real-stack register.*

Fast data entry will typically use the sequence Im(z) , ENTER^, Re(z), followed by the complex function. This is called the "Direct" data entry, as opposed to the "Natural" data entry, which would first input the real part. The 41Z module includes the function "**^IM/AG**" that can be used to input the number using the "Natural" convention (reversed from the Direct one).

Its usage is the same as the "i"-function on the HP-35s, to separate the real and the imaginary parts. The sequence is completed by pressing ENTER^ or R/S, after which the imaginary part will be left in the Y register and the real part in the X register as explained before.

(Incidentally, the 42S implementation of the complex stack isn't suitable for a true 4-level, since the COMPLEX function requires two levels prior to making the conversion!)

Other functions and special functionality in the 41Z module can be used as shortcuts to input purely real or imaginary numbers more efficiently. For instance, to enter the imaginary unit one need only press: 1, **ZIMAG^** (which is also equivalent to executing the **IMAGINE** function) – or simply **"ZKBRD, Radix, 1**" using the "complex keypad". And to enter 4 as a complex number, just press: 4, **ZREAL^ -** or simply "**ZKBRD, 4**" using the "complex keypad".

Incidentally, the 42S implementation fails short from delivering a true 4-level stack, due to the COMPLEX function and the fact that it requires two stack levels to be available to combine the complex number. In this regard the 41Z solution is a better one.

<==>

Two (opposite) alternatives to data entry: COMPLEX key on the 42S, and " ï" key on the 35S

3. User interface enhancements.

Table-3.1: Functions to enhance the user interface.

Index	Function	Group	Description
1	**ZK?YN**	Usability	Activates and deactivates the Complex Assignments
2	**ZKBRD**	Usability	Accesses most of the 41Z functions plus special features
3	**ZAVIEW**	Display	Views complex number in X,Y
4	**POLAR**	Display	Displays complex numbers in Polar form
5	**RECT**	Display	Displays complex numbers in Rectangular form
6	**^IM/AG**	Usability	Inputs Imaginary Part (or Argument) of complex number

These functions facilitate the showing of the complex number on the display, and the conversion between the polar and rectangular forms. They enhance the usability by supplying a system to handle the lack of native complex number treatment capabilities of the calculator.

3.1 Display mode and conversion functions.

ZAVIEW	**Complex number AVIEW**	Uses ALPHA registers	

Shows the contents of the complex stack level **Z** in the display, using the current complex display mode (POLAR or RECT).:

RECT: Re(*z*) + J Im(*z*) ; where Re(*z*) is stored in register X and Im(*z*) in register Y.
POLAR: Mod(z) <| Arg(z); *shown but not stored in the X,Y stack registers* (!)

Note that **ZAVIEW** uses the ALPHA register, thus the previous contents of the M, N and O registers will be lost.

The displaying will respect the current DEG, RAD, or GRAD angular mode (in POLAR form), the current FIX, SCI or ENG settings, as well as the number of decimal places selected on the calculator. Note that "J" precedes the imaginary part, as this improves legibility with real-life complex numbers, with decimal imaginary parts.

For a simplified visualization, **ZAVIEW** ***won't show decimal zeroes if the number is an integer***. This is done automatically regardless of the number of decimal places selected in the calculator; so one can immediately tell whether the real or imaginary parts are true integers as opposed to having some decimal content hidden in the least significant places not shown.

versus:


ZAVIEW ***will <u>extract common factor</u> if both the real and imaginary parts are equal:***

or also:


Executing the functions POLAR and RECT will also display the complex number currently stored in X,Y

POLAR	**Sets POLAR mode on**	Displays number	Shows in SET mode
RECT	**Sets RECT mode on**	Displays number	Shows in SET mode
ZPOL	**Convert to Polar**	Converts X,Y to POLAR	*Always shows in POLAR*
ZREC	**Convert to Rectangular**	Converts X,Y to RECT	Shows in SET mode

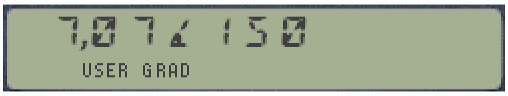
ZPOL Converts the complex number in the **Z** stack level from rectangular to polar mode. If executed in run mode, the display shows the value of its magnitude (its module) and its argument, as follows:

Mod < Arg ; where:
Mod = |z| and Arg=α **[*z* = |z| * e^i**α]

The argument value will be expressed in the angular settings currently selected: DEG, RAD, or GRAD.

equals

or also



ZREC is the reciprocal function, and will convert the complex number in **Z** (assumed to be in polar form) to rectangular form, showing it on the display (in run mode) in identical manner as **ZAVIEW**.

In fact, if it weren't because of the displaying capabilities, these two functions will be identical to the pair R-P and P-R, standard on the calculator. Recognizing this, they're assigned to the very same position as their real counterparts on the Complex User keyboard.

Notice that contrary to the **POLAR** and **RECT** functions (which only display the values), **ZPOL** and **ZREC** *perform the actual conversion of the values and store them in the stack registers* (complex and real). This is also very useful to enter complex numbers directly in polar form, simply using the sequence: (direct data entry: Angle first, then modulus):

- Arg(z), ENTER^, |z|, **ZREC** -> Re(z) + J Im(z)

3.2 Complex Natural Data Entry.

This function belongs to its own category, as an automated way to input a complex number using the "Natural" data entry convention: Real part first, Imaginary part next. Its major advantage (besides allowing the natural data entry sequence) is that *it performs a complex stack lift upon completion of the data entry*, thus there's no need to use **ZENTER^** to input the complex number into the complex stack. That alone justifies its inclusion on the 41Z module.

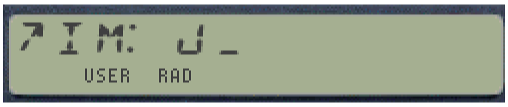
^IM/AG _	**Inputs Im(z)/Arg(z) Part**	**Does Stack Lift**	Prompting function

The function will prompt for the imaginary part (or the argument if in POLAR mode) of the complex number being entered. The design mimics that on the HP-35S calculator, and it's used as a way to separate the two complex parts during the complex number data entering.

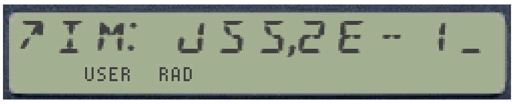
A few important considerations are:

- The real part (or module) must be introduced right *before* calling it, so it's in X during the data entry.
- The keyboard is redefined to allow for numeric digits, RADIX, CHS and EEX as only alid keys.
- The radix symbol used (comma or dot) is controlled by the user flag 28.
- Only one RADIX character will be allowed in the mantissa – and none in the exponent.
- Only nine digits will be used for the mantissa, and two in the exponent. **^IM/AG** will not check for that during the input process, but exceeding entries will simply be ignored.
- Only one EEX can exist in the imaginary part - **^IM/AG** will check for that.
- Only one CHS can be used for the mantissa sign, **^IM/AG** will check for that.
- Multiple CHS can be used for the exponent sign, but **^IM/AG** will apply the arithmetic rules to determine the final sign as follows: odd number is negative, even number is positive.
- Pressing Back Arrow will remove the last entry, be that a number, Radix, EEX or CHS. If the entry is the first one it will cancel the process and will discard the real part as well.
- The sequence must be ended by pressing ENTER^ or R/S.
- The display cue is different depending on the actual complex mode (RECT or POLAR), and it's controlled automatically.
- Upon completion, the complex number is pushed into the **Z** complex stack level, and placed on the X,Y real stack registers as well following the same 41Z convention: real part in X and imaginary part in Y. The complex stack is lifted and the real stack is synchronized accordingly.

The screens below show usage examples in RECT and POLAR modes:



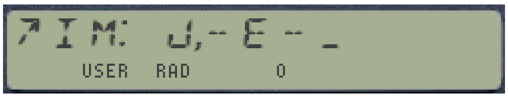
until finally:



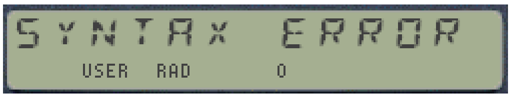
ending as:

Note: To extract the numeric value from the input string, **^IM/AG** executes the same code as the X-function **ANUM**. All conversion conventions will follow the same **ANUM** logic. Suffice it to say that the implementation of **^IM/AG** is not absolute perfect and you can trip it up if that's what you really want – but it should prevent likely errors that could yield incorrect results. It's a very convenient way to meet this need solving the diverse issues associated with its generic character.

If the input string doesn't yield any sensible numeric result, the message "SYNTAX ERROR" is briefly shown in the display, and the stack is restored to its status prior to executing **^IM/AG**.

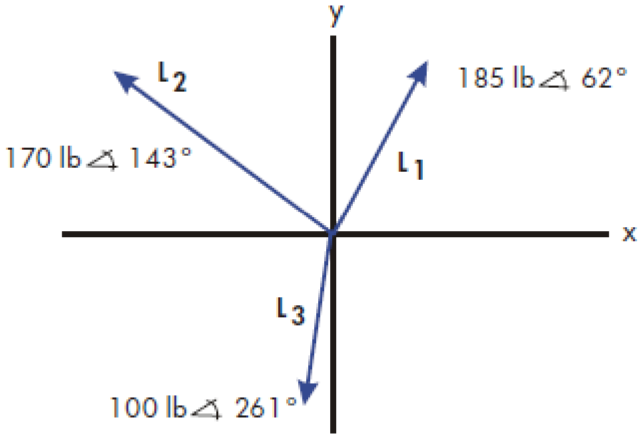


will trigger:



Some apparently incorrect syntax constructions will however be properly interpreted by **^IM/AG**, returning a valid imaginary part. This is for instance the case with multiple negative signs in the exponent, or decimal values after negative sign in the mantissa. Such is the flexibility of the ANUM function!

Example: *Vector Load addition* (taken from the 35s User Guide):-



We start by setting POLAR and DEG modes, then using the ^IM/AG function three times will set the three complex numbers on the complex stack, and finally simply execute the complex addition function Z+ twice:

POLAR, DEG
185**,** **^IM/AG**, 62**,** ENTER^
170, **^IM/AG**, 143, R/S
100, **^IM/AG**, 261, R/S
Z+, Z+

Result: **-> 178,9372 <) 111,1489**

Or in Rectangular mode (as it's saved in XY): **RECT** -> -64,559 + J166,885

Note the following points:

- We used indistinctly ENTER^ and R/S to terminate the complex number entry.
- No need to store intermediate results as the complex buffer can hold up to four levels.
- We didn't need to use **ZENTER^** to push the complex numbers into the complex stack because the stack-lift was performed by **^IM/AG**.

With regard to the data entry sequence, one could have used **ZREC** instead of **^IM/AG** – albeit in that case it would have been in "direct mode", as opposed to the more intuitive natural convention. It also requires pressing **ZENTER^** to push each number into the complex stack.

This is the keystroke sequence and partial results (assuming we're in **POLAR** mode)

62, ENTER^, 185, **ZREC**, **ZENTER^**	-> 185 <)62
143, ENTER^, 170, **ZREC**, **ZENTER^**	-> 170 <)143
261, ENTER^, 100, **ZREC**	-> 100 <)-99
Z+, Z+	-> 178,9372 <) 111,1489

One last remark about data displaying vs. data entry.- As it was explained before, **ZPOL** will convert the complex number into Polar coordinates, and it will be displayed in **POLAR** form even if **RECT** mode is selected. This is the single one exception all throughout the 41z module, and it will only work immediately after pressing **ZPOL** but not for subsequent executions of **ZAVIEW** – which always expects the number is stored in rectangular form, and therefore will show an incorrect expression.

3.3 The Complex User Assignments.

The 41Z module provides a convenient way to do user key assignments *in masse*. Given the parallelisms between the real and complex number functions, the natural choice for many of the functions is "predetermined" to be that of their real counterparts.

A single function is used for the mass-assignment (or de-assignment) action:

ZK?YN	Complex User Assignments		Prompting function

ZK?YN automates the assignment and de-assignment of 37 functions. It prompts for a Yes/No answer, as follows:

- Answering "Y" will assign the complex functions to their target keys
- Answering "N" will de-assign them, and
- Pressing "Back Arrow" will cancel the function.
- Any other key input (including ON) will be ignored.

The assignment action will be indicated by the message "Z-KEYS: ON" or "Z-KEYS OFF" in the display during the time it takes to perform, followed by "PACKING" – and possibly "TRY AGAIN" should the enough number of memory registers not exist.

Note that **ZK?YN** is *selective*: any other key assignment not part of the complex functions set will not be modified.

Keycode	Unshifted Keys		Shifted Keys	
11	s+	ZHYP	s-	ZNXT
12	1/X	ZINV	y^x	W^Z
13	SQRT	ZSQRT	x^2	Z^2
14	LOG	ZLOG	10^x	ZALOG
15	LN	ZLN	e^x	ZEXP
21	x<>y	Z<>W	CLs	ZTRP
22	RDN	ZRDN	%	ZRUP
23	SIN	ZSIN	ASIN	ZASIN
24	COS	ZCOS	ACOS	ZACOS
25	TAN	ZTAN	ATAN	ZATAN
33	STO	ZSTO	LBL	n/a
34	RCL	ZRCL	GTO	n/a
41	ENTER^	^IMG	CAT	ZENTER^
42	CHS	n/a	ISG	ZNEG
44	EEX	n/a	CLx	CLZ
51	-	Z-	x=y?	Z=W?
61	+	Z+	x<=y?	Z=WR?
71	*	Z*	x>y?	Z=I?
81	/	Z/	x=0?	Z=0?
83	,	n/a	LASTx	LASTZ
84	R/S	ZAVIEW	VIEW	ZVIEW

Table 3.3. Complex key assignments done by ZK?YN

3.4 The Complex Keyboard.

As good as the user assignments are to effectively map out many of the 41Z functions, this method is not free from inconveniences. Perhaps the biggest disadvantage of the Complex Assignments is that it's frequently required to toggle the user mode back and forth, depending on whether it's a complex or a real (native) function to be executed.

Besides that, the Complex Assignments consume a relative large number of memory registers that can be needed for other purposes. Lastly, there are numerous 41Z functions not included on the user assignments map, and no more "logical" keys are available without compromising the usability of the calculator.
To solve these quibbles, the 41Z module provides an alternative method to access the majority of the complex functions, plus some unique additional functionality. It's called the ***Complex Keyboard***, accessed by the function **ZKBRD**: a single key assignment unleashes the complete potential of the module, used as a ***complex prefix***, or in different combinations with the SHIFT key and with itself.

Figure 3.4. Complex Keyboard overlay (with ZKBRD assigned to Sigma+).
On the left: the version for V41. On the right, for i41CX

© 2009 M. Luján García.

Here's how to access all the functions using **ZKBRD**:

*a.- **Direct functions***. Simply press "**Z**" as a prefix to denote that the next function will operate on a complex argument, and not on a real one. These functions don't have any special marks, as they correspond to the standard functions on the HP-41 keyboard.

Examples: Pressing **Z**, LN will execute **ZLN**; pressing **Z**, COS will execute **ZCOS**, etc...
Pressing **Z**, + will execute **Z+**; pressing **Z**, R/S will execute **ZAVIEW**,

There are *twenty 41Z functions* directly accessible like these.

b.- Shifted functions. Press "**Z**" followed by the SHIFT key. These functions are either marked in blue when different from the standard SHIFTED ones, or just marked in yellow as part of the standard HP-41 keyboard (like x=y?, which will execute **Z=W?** if the pressed key sequence is this: **Z**, SHIFT, x=y?

Examples: pressing **Z**, SHIFT, LN will execute **ZEXP**; pressing **Z**, SHIFT, SIN will execute **ZASIN**,

Pressing **Z**, SHIFT, R/S will execute **ZVIEW** (a prompting function itself).

There are *thirty-one 41Z functions* accessible using this SHIFTED method.

c.- Dual (alternate) functions. Press "**Z**" *twice* as a double prefix to access the dual complex functions and many others. These functions are marked in red, on the right side of each available key.

Examples: Pressing **Z, Z**, 7 will execute **ZWDET**; pressing **Z, Z**, 5 will execute **ZWCROSS**, , and so on with all the "red-labeled" keys.

Pressing **Z, Z**, ENTER^ will execute **ZREPL**; pressing **Z, Z, Z** will execute **Z<>U**

There are *twenty-five 41Z functions* accessible using this Dual method.

d.- Multi-value functions. As a particular case of the dual functions case above, the ZNEXT function group is enabled by pressing "**Z**" *twice* and then SHIFT. This group is encircled on the keyboard overlay, and sets the five multi-value functions as follows: **NXTASN, NXTACS, NXTATN, NXTLN**, and **NXTNRT** (this one will also prompt for the root order, as an integer number 0-9).

Notice that pressing SHIFT while in the NEXT section toggles the display to "ZBSL". Use it as a shortcut to access the different Bessel functions of first and second kind provided in the 41, as follows: **ZJBS**, **ZIBS**, **ZKBS**, and **ZYBS**. – as well as **EIZ/IZ**, a particular case of Spherical Hankel h1(0,z).

e.- Hyperbolic functions. Press "**Z**" followed by SHIFT *twice* to access the three direct hyperbolics. Pressing SHIFT a third time will add the letter "A" to the function name and will enable the inverse functions. This action toggles with each subsequent pressing of SHIFT. (Watch the 41Z building up the function name in the display as you press the keys...)

Example: Pressing **Z**, SHIFT, SHIFT, SHIFT, **SIN** will execute **ZASINH**

f.- Complex Keypads. Press "**Z**" followed by a numeric key (0 to 9) to enter the corresponding digit as a complex number in the complex stack. Pressing "**Z**" followed by the Radix key, and then the numeric key will input the digit as an imaginary number as opposed to as a real number into the complex stack. This is a very useful shortcut to quickly input integer real or imaginary values for complex arithmetic or other operations (like multiplying by 2, etc.)

Pressing **Z**, XEQ calls the function **^IM/AG** for the Natural Data entry. This is obviously not shown on the keyboard - which has no changes to the key legends for un-shifted functions. Note that there are three different ways to invoke **^IM/AG**, as follows:

XEQ, ALPHA, SHIFT, N, I, M, /, A, G, ALPHA	-> the standard HP-41 method, or:
Z, SHIFT, ENTER^	-> shown in blue in the overlay, or:
Z, XEQ	-> not shown.

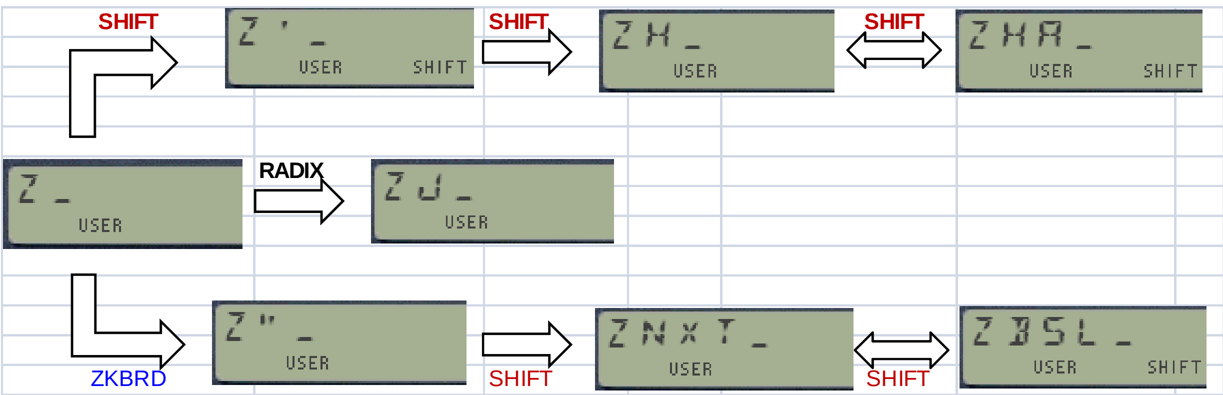
Other keystrokes. The 41Z module takes control of the calculator keyboard when **ZKBRD** is executed. Available keys are determined by the partial key sequence entered, as defined on the 41Z Keys overlay and as explained above. Pressing **USER** or **ALPHA** will have no effect, and pressing **ON** at any time will shut the calculator off. The *back arrow* key plays its usual important role during data entering, and also undoes the last key pressed during a multi-shifted key sequence. Try it by yourself and you'll see it's actually easier than giving examples on how it works here.

In summary: a complete new keyboard that is accessed by the "**Z**" blue prefix key. This being the only requisite, it's a near-perfect compromise once you get used to it - but if you don't like it you can use the User Assignments , the choice is yours.

The 41Z overlay can be downloaded from the HP-41 archive website, at:
http://www.hp41.org/LibView.cfm?Command=View&ItemID=893

To use it with V41 emulator, replace the original file "*large.bmp*" in the V41 directory with the 41Z bitmap file, after renaming it to the same file name.

The figure below shows the main different modes of the **ZKBRD** function, the real cornerstone of the 41Z module:



Press the Back-arrow key to bring the command chain back to the starting point (ZKBRD). Pressing it twice shows "NULL" and cancels out the sequence.

Pressing non-relevant keys (i.e. those not supposed to be included in the corresponding mode) causes the display to blink, and maintain the same prompt (no action taken).

4. Stack and Memory functions.

Let **Z** and **W** be the lower two levels of the complex stack, and *z* and *w* two complex numbers stored in **Z** and **W** respectively. **Z** = Re(z)+ j Im(z); **W** = Re(w) + j Im(w)

Note the use of "j" to express the imaginary unit, instead of "i" . This isn't done to favor those EE's in the audience (you know who *we* are), but rather due to the displaying limitations of the 41 display: no lower-case letters for either i or j, and better-looking for the last one in caps.

Note also that despite their being used interchangeably, the complex stack register "**Z**" – in bold font - and the real stack register "Z" – in regular font - are not the same at all.

Table-4.1: Stack and memory function group.

Index	Function	Name	Description
1	**ZTRP**	Re(z)<>Im(z)	Exchanges (transposes) Re and Im for number in level **Z**.
2	**ZENTER^**	Complex ENTER^	Enters X,Y into complex level **Z, lifts complex stack.**
3	**ZREPL**	Complex Stack Fill	Fills complex stack with value(s) in X,Y
4	**ZRDN**	Complex Roll Down	Rolls complex stack down
5	**ZRUP**	Complex Roll Up	Rolls complex stack up
6	**ZREAL^**	Inputs real Z	Enters value in X as real-part only complex number
7	**ZIMAG^**	Inputs imaginary Z	Enters value in X as imaginary complex number
8	**Z<>W**	Complex Z<>W	Swaps complex levels **Z** and **W**
9 (*)	**Z<>ST _ _**	Complex Z<> level	Swaps complex levels **Z** and any stack level (0-4)
10 (*)	**ZRCL _ _**	Complex Recall	Recalls complex number from memory to level **Z**
11 (*)	**ZSTO _ _**	Complex Storage	Stores complex number in **Z** into memory
12 (*)	**Z<> _ _**	Complex Exchange	Exchanges number in level **Z** and memory
13 (*)	**ZVIEW _ _**	Complex Display	Shows Complex number stored in memory register
14	**CLZ**	Clears Level Z	Deletes complex level **Z**
15	**CLZST**	Clears Complex Stack	Clears all complex levels **U, V, W**, and **Z**
16	**ZREAL**	Extracts real part	Clears Im(z)
17	**ZIMAG**	Extracts Imag part	Clears Re(z)
18	**LASTZ**	Last number used	Recovers the last complex number used

*(*) Note: These functions are **fully programmable**. When used in a program their argument is taken from the next program line, see below for details.*

4.1 Stack and memory functions group.

Let's start with the individual description of these functions in more detail, beginning with the simplest.

ZTRP	**Z Transpose**	Does Re <>Im	

This function's very modest goal is to exchange the real and imaginary parts of the complex number stored in the **Z** level of the complex stack.

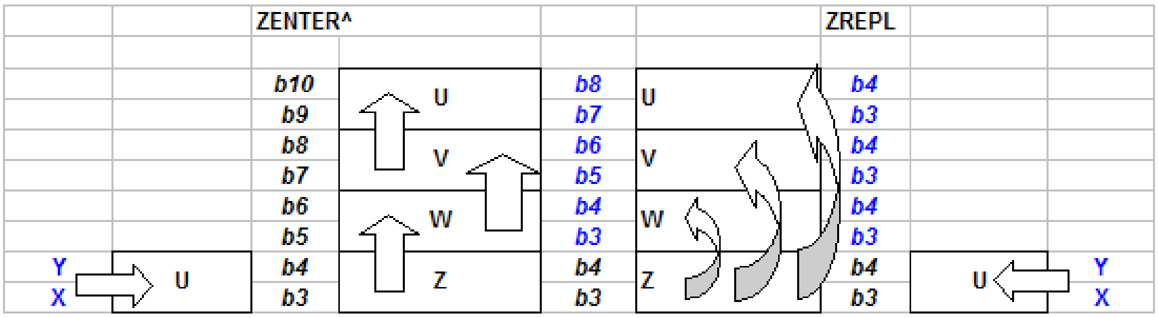
Hardly a worthwhile scope, you'd say, considering that the standard function X<>Y does the same thing? Indeed it is quite similar (and as such it's logically assigned to the shifted X<>Y key). But it's not quite the same, as in run mode **ZTRP** also shows on the display the complex number after transposing their real and imaginary parts. Besides, as it was mentioned in the introduction, this

function may play an important role during data entry: it is the one to use when entering the real part first, as per the following sequence: Re(*z*), ENTER^, Im(*z*), **ZTRP**

Thus its use is analogous to the "COMPLEX" function on the HP-42S, also required to enter the complex number in the stack, from its two real components. Note that the other, alternative data entering sequence doesn't require using ZTRP, although the order of the real and imaginary parts is reversed (and arguably less intuitive): Im(*z*), ENTER^, Re(*z*). Either one of these two is entirely adequate once you become familiar with it and get used to using it - it's your choice.

ZENTER^	**Enters X,Y into levels Z, W**	**Does Stack lift**	
ZRPL^	**Fills complex stack**		

ZENTER^ enters the values in X,Y as a complex number in the **Z** stack level, and performs stack lift (thus duplicates **Z** into **W** as well – and **U** is lost due to the complex stack spill-over). As said in the introduction, *always* use **ZENTER^** to perform stack lift when entering two (or more) complex numbers into the complex stack. This is required for the correct operation of dual complex functions, like **Z+**, or when doing chain calculations using the complex stack (which, unlike the real XYZT real stack, it does NOT have an automated stack lift triggered by the introduction of a new real number).



ZREPL simply fills the complex stack with the values in the real registers X,Y. This is convenient in chained calculations (like the Horner method for polynomial evaluation). If executed in run mode it also displays the number in **Z**. This is in fact a common characteristic of all the functions in the 41Z module, built so to provide visual feedback of the action performed.

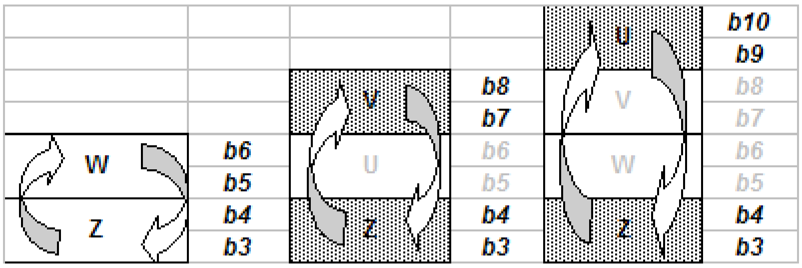
ZREAL^	**Enters X in Z as (x+j0)**	**Does Stack Lift**	
ZIMAG^	**Enters X in Z as (0+jX)**	**Does Stack Lift**	

These functions enter the value in X either as a purely real or purely imaginary number in complex form in the **Z** stack level, and perform stack lift. If executed in run mode it also displays the number in **Z** upon completion.

Z<>ST (*)	**Exchanges Z and Stack**	Level# = 0,1,2,3,4	Prompting function
Z<>V	**Exchanges Z and V**		
Z<>W	**Exchanges Z and W**		

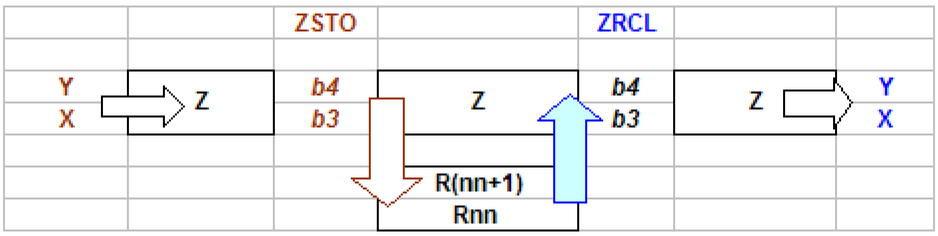
(*) Fully programmable, see note in previous page.

Use these functions to swap the contents of the **Z** and **U**/**V**/**W** levels of the complex stack respectively. As always, the execution ends with **ZAVIEW** in run mode, displaying the new contents of the **Z** register.(which is also copied into the XY registers).



ZRCL _ _	**Recall from Complex Register**	**Does Stack lift**	Prompting function
ZSTO _ _	**Store in Complex Register**		Prompting function

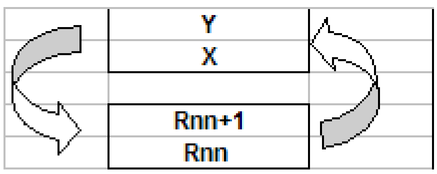
Like their real counterparts, these functions are used to Recall or store the complex number in Z from or into the complex register which number is specified as the function's argument. In fact two (real) storage registers are used, one for the imaginary part and another for the real part. This means that CRnn corresponds to the real storage registers Rnn and R(nn+1).



ZRCL will perform complex stack lift upon recalling the contents of the memory registers to the Z stack level. Also note that, following the 41Z convention, **ZSTO** will overwrite the Z level with the contents of X,Y if these were not the same. This allows walk-up complex data entering.

ZVIEW _ _	**Displays Complex Register value**		Prompting function
Z<> _ _	**Exchanges Z and complex register**		Prompting function

Like its real counterparts, these functions view or exchange the content of the complex stack level **Z** with that of the complex storage register given as its argument. Two standard storage registers are used, as per the above description.



These four functions are ***fully programmable***. When in program mode (either running or SST execution), the index input is ignored, and their argument is taken from the following program line after the function. For this reason they are sometimes called *non-merged* functions. In fact, the number denoting the argument can have any combination of leading zeroes (like 001, 01, 1 all resulting in the same). Moreover, when the argument is zero then such index following line can be omitted if any non-numeric line follows the function. This saves bytes. This implementation was

written by W. Doug Wilder, and it is even more convenient than the one used by the HEPAX module for its own multi-function groups.

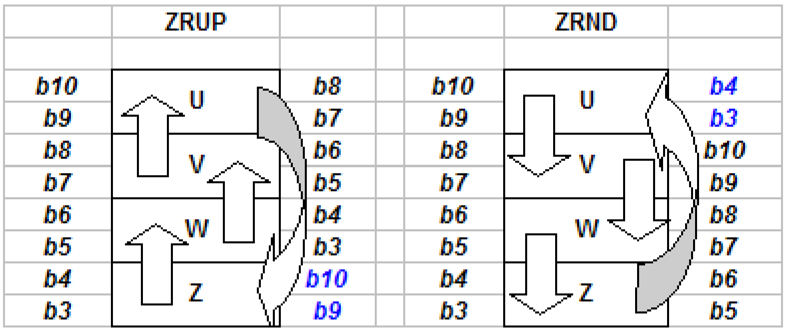
Similar to the real counterparts, keys on the first two rows can be used as ***shortcut for indexes 1-9***. Note that ***indirect addressing is also supported*** (say **ZRCL IND _ _**) pressing the SHIFT key - *in RUN mode only* (i.e. not programmatically). In program mode you can make use of the fact that the ***indirect addressing is nothing more that adding 128 to the address***, thus it can be handled by simply adding such factor to the index in the second program line.

Also note that despite being possible to invoke, their logic doesn't support the use of the stack registers. (**ZRCL ST _**); and certainly neither the combination of both, indirect and stack addressing (**ZRCL IND ST _ _**). If you use these, unpredicted (and wrong) results will occur. The same can be said if you press the arithmetic keys (**+, -, *, /**): **simply *don't*.**

Lastly, a NONEXISTENT message will be shown if the storage register is not available in main memory. Registers can be made available using the SIZE function of the calculator.

ZRDN	**Rolls complex stack down**		
ZRUP	**Rolls complex stack up**		

Like their real stack counterparts, these functions will roll the complex stack down or up respectively. If executed in run mode it also displays the number in **Z**. Real stack registers will be synchronized accordingly.



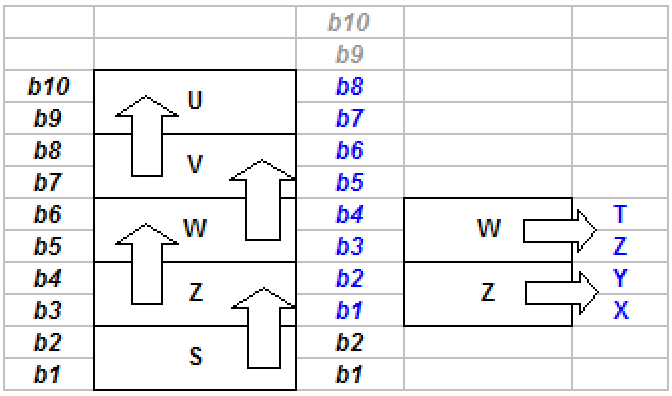
Be aware that although **ZRDN** and **ZRUP** do not perform stack lift, they update the Z complex register with the values present in X,Y upon the function execution. This behavior is common across all 41Z functions.

CLZ	**Clears complex stack level Z**		
CLZST	**Clears complete complex stack**		
ZREAL	**Extracts Real part from Z**		
ZIMAG	**Extracts Imaginary part from Z**		

Use these four functions to partially or completely clear (delete) the contents of the complex stack **Z** level, or the complete complex stack. No frills, no caveats. The real stack will also be cleared appropriately.

LASTZ	Recalls last number used to Z	Does Stack Lift	

Similar to the LASTX function, **LASTZ** recalls the number used in the immediate preceding operation back to the **Z** level of the complex stack. A complex stack lift is performed, pushing the contents of **Z** up to the level **W**, and losing the previous content of **U**.



The majority of functions on the 41Z module perform an automated storage of their argument into the LastZ register, enabling the subsequent using of **LASTZ**. This will be notated in this manual when appropriate under each function description.

Example: to calculate $[(z^2 + z)/2]$ simply press: **ZSQRT, LASTZ, Z+, ZHALF**

Example: Calculate the following expression without using any data registers:

F(z) = Ln [z + SQR(z^2 + 1)], for z= 20+20i

Solution:

20, ENTER^, **ZRPL**	-> puts 20+20i in all 4 levels of the complex stack
Z^2, 1, **ZREAL^**, **Z+**	-> could have used "1, +" as a more direct method
ZSQRT, **Z+**, **ZLN**	**-> 4,035+J0,785**

Congratulations! You just calculated the hyperbolic arcsine of (20+20i).

4.2. ZSTO Math function group.

ZST+ _ _	Recall from Complex Register		Prompting function
ZST- _ _	Store in Complex Register		Prompting function
ZST* _ _	Recall from Complex Register		Prompting function
ZST/ _ _	Store in Complex Register		Prompting function

The newest addition to the 41Z function set.- allow storage math in a concise format, saving bytes and programming steps in FOCAL programs. Their equivalence with standard functions would have to be done using four steps, and disturbing the Complex Stack as follows:

1.- ZENTER^, 2.- Z<>(nn) 3.- MATH (+, -, *, /) 3.- Z<>(nn)

Functions are fully programmable using the non-merged technique. These functions can be accessed using the Z-keyboard from its own dedicated launcher, pressing „Z“ twice and then „STO“.

5. Complex Math.

Complex numbers are much more than a simple extension of the real numbers into two dimensions. The Complex Plane is a mathematical domain with well-defined, own properties and singularities, and it isn't in the scope of this manual to treat all its fundamental properties. On occasions there will be a short discussion for a few functions (notably the logarithms!), and some analogies will be made to their geometric equivalences, but it is assumed throughout this manual that the user has a good understanding of complex numbers and their properties.

5.1. Arithmetic and Simple Math.

Table-5.1:- Arithmetic functions.

Index	Function	Formula	Description
1	**Z+**	Z=w+z	Complex addition
2	**Z-**	Z=w-z	Complex subtraction
3	**Z***	Z=w*z	Complex multiplication
4	**Z/**	Z=w/z	Complex division
5a	**ZINV**	Z=1/z	Complex inversion, direct formula
5b	**1/Z**	Z=1/r e^(-iArg)	Complex inversion, uses TOPOL
6	**ZDBL**	z=2*z	Doubles the complex number
7	**ZHALF**	z= z/2	Halves the complex number
8	**ZRND**	Z=rounded(z)	Rounds Z to display settings precision
9	**ZINT**	Z=Int(z)	Takes integer part for Re(z) and Im(z)
10	**ZFRC**	Z=Frc(z)	Takes fractional part for Re(z) and Im(z)
11	**ZPIX**	Z=zπ	Simple multiplication by pi

Here's a description of the individual functions within this group.

Z+	**Complex addition**	Z=w+z	Does LastZ
Z-	**Complex subtraction**	Z=w-z	Does LastZ
Z*	**Complex multiplication**	Z=w*z	Does LastZ
Z/	**Complex division**	Z=w/z	Does LastZ

Complex arithmetic using the RPN scheme, with the first number stored in the **W** stack level and the second in the **Z** stack level. The result is stored in the **Z** level, the complex stack drops (duplicating **U** into **V**), and the previous contents of **Z** is saved in the LastZ register.

ZINV	**Direct Complex inversion**	Z=1/z	Does LastZ
1/Z	**Uses POLAR conversion**	Z=1/r e^(-iArg)	Does LastZ

Calculates the reciprocal of the complex number stored in **Z**. The result is saved in **Z** and the original argument saved in the LastZ register. Of these two the direct method is faster and of comparable accuracy – thus it's the preferred one, as well as the one used as subroutine for other functions.

This function would be equivalent to a particular case of **Z/**, where w=1+0j, and not using the stack level **W**. Note however that **Z/** implementation is not based on the **ZINV** algorithm [that is, making use of the fact that : w/z = w * (1/z)], but based directly on the real and imaginary parts of both arguments.

Example. Calculate z/z using **ZINV** for z=i

We'll use the direct data entry, starting w/ the imaginary part:

1, ENTER^, 0, **ZINV**	-> 0-j1
LASTZ	-> 0+j1
Z*	-> 1+j0

Note that *integer numbers are displayed without decimal zeroes,* simplifying the visual display of the complex numbers.

ZDBL	**Doubles Z**	Z=2*z	Does LastZ
ZHALF	**Halves Z**	Z=z/2	Does LastZ

These two functions are provided to save stack level usage and programming efficiency. The same result can also be accomplished using their generic forms (like **Z*** and **Z/**, with *w*=2+0j), but the shortcuts are faster and simpler to use.

Example. Taken from the HP-41 Advantage manual, page 97.

Calculate: $z_1/(z_2+z_3)$; for: z_1=(23+13i;) z_2=(-2+i), and z_3=(4-3i)

If the complex stack were limited to 2 levels deep, we would need to calculate the inverse of the denominator and multiply it by the numerator, but using the 4-level deep complex stack there's no need to resort to that workaround. We can do as follows:

13, ENTER, 23, **ZENTER^**	-> 23+j13
1, ENTER^, 2, CHS, **ZENTER**^	-> -2+j1
3, CHS, ENTER^, 4, **Z+**	-> 2(1-j)
Z/	-> 2,500+j9

Note that 41Z *automatically takes common factor when appropriate*, and that integer numbers are displayed without decimal zeroes to simplify the visuals display of the complex numbers. Non-integers are displayed using the current decimal settings, but of course full precision (that is 9 decimal places) is always used for the calculations (except in the rounding functions).

ZRND	**Rounds Complex number**	Z=Rounded(z)	Does LastZ
ZINT	**Takes integer parts**	Z=Int[Re(z)+jInt[Im(z)	Does LastZ
ZFRC	**Takes Fractional parts**	Z=Frc[Re(z)+jFrc[Im(z)	Does LastZ

These functions will round, take integer part or fractional part both the real and imaginary parts of the complex number in **Z.** The rounding is done according to the current decimal places specified by the display settings.

ZPIX	**Multiplies by pi**	Z=π*z	Does LastZ

Simple multiplication by pi, used as a shortcut in the Bessel FOCAL programs. Has better accuracy than the FOCAL method, as it used internal 13-digit math.

5.2. Exponential and powers that be.

Table-5.2: Exponential group.

Index	Function	Formula	Description
1a	**ZEXP**	Z=REC(e^x, y)	Complex exponential (method one)
1b	**e^Z**	See below	Complex Exponential (method two)
2	**Z^2**	Z=REC(r^2, 2α)	Complex square
3a	**ZSQRT**	Algebraic Formula	Principal value of complex square root
3b	**SQRTZ**	Z=REC(r^1/2, α/2)	Principal value of complex square root
4	**W^Z**	Z=e^z*Ln(w)	Complex to complex Power
5	**W^1/Z**	Z=e^1/z*Ln(w)	Complex to reciprocal complex Power
6	**X^Z**	Z=e^z*Ln(x)	Real to complex power
7	**X^1/Z**	Z=e^z*Ln(x)	Real to reciprocal complex power
8	**Z^X**	Z=e^x*Ln(z)	Complex to real Power
9	**Z^1/X**	Z=e^1/x*Ln(z)	Complex to reciprocal real Power
10	**ZALOG**	Z=e^z*Ln(10)	Complex decimal power
11	**NXTRTN**	Z=z*e^j 2π/N	Next value of complex nth. Root

Looking at the above formula table it's easy to realize the importance of the exponential and logarithmic functions, as they are used to derive many of the other functions in the 41Z module. It is therefore important to define them properly and implement them in an efficient way.

The 41Z module includes two different ways to calculate the complex exponential function. The first one is based on the trigonometric expressions, and the second one uses the built-in polar to rectangular routines, which have enough precision in the majority of practical cases. The first method is slightly more precise but takes longer computation time.

ZEXP	**Complex Exponential**	Z=REC(e^x, y)	Does LastZ
e^Z	**Complex Exponential**	Trigonometric	Does LastZ

One could have used the rectangular expressions to calculate the result, as follows:

e^z = e^x * (cos y + i sin y), thus: Re(z) = e^(x) * cos y ; and: Im(z) = e^(x) * sin y

and this is how the function **e^Z** has been programmed. It is however more efficient (albeit slightly less precise) to work in polar form, as follows:

since z= x+iy, then e^z = e^(x+iy) = e^x * e^iy,

and to calculate the final result we only need to convert the above number to rectangular form.

Example.- Calculate exp(z^{-2}), for z=(1+i)

1, ENTER^, **ZENTER^**	-> 1(1+j)
2, CHS, **Z^X**	-> 0-j0,500
ZEXP	-> 0,878-j0,479

Another method using **W^Z** and the complex keypad function (**ZREAL^**):

1, ENTER^, **ZENTER^**	-> 1(1+j)
2, CHS, **ZREAL^**	-> -2+j0
W^Z, ZEXP	-> 0,878-j0,479

or alternatively, this shorter and more efficient way: (leaves **W** undisturbed)

1, ENTER^, **Z^2, ZINV, ZEXP** -> 0,878-j0,479

Note how this last method doesn't require using **ZENTER^** to terminate the data input sequence, as the execution of monadic functions will automatically synchronize the complex stack level Z with the contents of the real X,Y registers.

Z^2	**Complex square**	Z=REC(r^2, 2α)	Does LastZ
ZSQRT	**Complex square root**	Algebraic Formula	Does LastZ
SQRTZ	**Complex square root**	Z=REC(r^1/2, α/2)	Does LastZ

Two particular cases also where working in polar form yields more effective handling. Consider that:

Z^2 = |z|^2 * e^2iα, and:
Sqrt(z) = z^1/2 = Sqrt(|z|) * e^iα, where α=Arg(z),

It is then simpler first converting the complex number to its polar form, and then apply the individual operations upon its constituents, followed by a final conversion back to the rectangular form.

Note that this implementation of **ZSQRT** only offers one of the two existing values for the square root of a given complex number. The other value is easily obtained as its opposite, thus the sum of both square roots is always zero.

Such isn't exclusive to complex arguments, for the same occurs in the real domain - where there are always 2 values, x1 and –x1, that satisfy the equation SQRT[(x1)^2].

As with other multi-valued functions, the returned value is called the *principal value* of the function. See section 6 ahead for a more extensive treatment of this problem.

W^Z	**Complex to complex Power**	Z=e^[z*Ln(w)]	Does LastZ
W^1/Z	**Complex to reciprocal Power**	Z=e^[Ln(w)/z]	Does LastZ

The most generic form of all power functions, calculated using the expressions:

w^z = exp[z*Ln(w)], and
w^1/z = exp[Ln(w) / z]

The second function is a more convenient way to handle the reciprocal power, but it's obviously identical to the combination **ZINV, W^Z**.

Example: calculate the inverse of the complex number 1+2i using **W^Z**:- Then obtain its reciprocal using **ZINV** to verify the calculations.

2, ENTER^, 1, **ZENTER^** number stored in level **W** (also as: 1, ENTER^, 2, **ZTRP**)
0, ENTER^, -1 exponent –1 stored in level **Z** (also as: -1, ENTER^, 0, **ZTRP**)
W^Z result: 0,200-j0,400
ZINV result: 1,000+j2

Note that the final result isn't exact – as the decimal zeroes in the real part indicate there's a loss of precision in the calculations.

Z^X	**Complex to real power**	Z=e^[x*Ln(z)]	Does LastZ
Z^1/X	**Complex to reciprocal real**	Z=e^[Ln(z)/x]	Does LastZ
X^Z	**Real to complex power**	Z=e^[z*Ln(x)]	Does LastZ
X^1/Z	**Real to reciprocal complex**	Z=e^[1/z*Ln(x)]	Does LastZ
ZALOG	**10 to complex power**	Z=e^[z*Ln(10)]	Does LastZ

These five functions are calculated as particular examples of the generic case W^Z. Their advantage is a faster data entry (not requiring inputting the zero value) and a better accuracy in the results

Z^1/X is identical to: 1/X, **Z^X**
X^1/Z is identical to: RDN, **ZINV**, R^, **X^Z**

Data entry is different for hybrid functions, with mixed complex and real arguments. As a rule, the second argument is stored into its corresponding stack register, as follows:

- x into the real stack register X for **Z^X** and **Z^1/X**
- *z* into the complex stack register **Z** for **X^Z** and **X^1/Z**

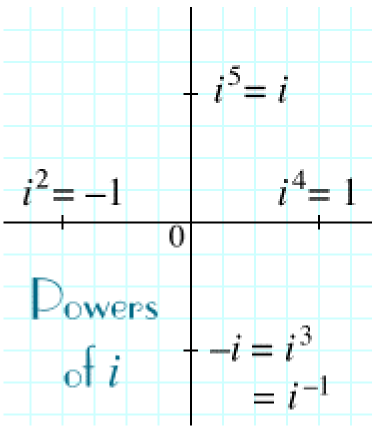
The first argument needs to be input first, since this is an RPN implementation.

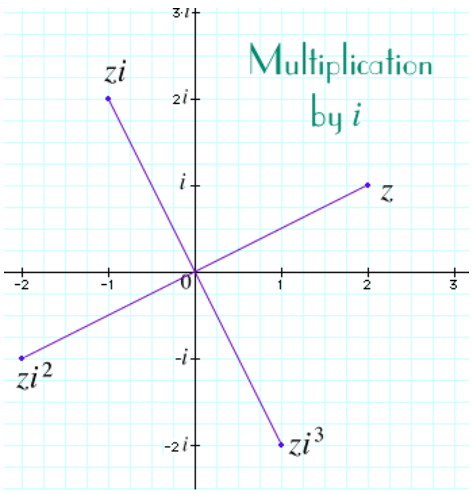
Because **ZALOG** is a monadic function, it expects z in the stack level **Z**, and thus it doesn't disturb the complex stack.

Example: Calculate (1+2i)^3 and 3^(1+2i)

2, ENTER^, 1, **ZENTER^**, 3, **Z^X** results: (1+2i)^3 = -11-2 j
2, ENTER^, 1, **ZENTER^**, 3, **X^Z** results: 3^(1+2i) = 1+0 j

Example: Verify the powers of the imaginary unit, as per the picture below.- You can use either **Z^X**, with z=(0+i) and x=1,2,3,4,5; or alternatively **W^Z**, with w=(0+i) and z=(1+0i), (2+0i), (3+0i), etc.





This keystroke sequence will quickly address the even powers:

0, ENTER^, 1, **ZTRP**	-> 0 + j1	i
Z^2	-> -1 + j0	i^2 = -1
Z^2	-> 1 + J0	i^4 = 1

Whilst this will take care of the rest (and also in general):

0, ENTER^, 1, **ZTRP**	->	0 + j1	i
3, **Z^X**	->	0 - j1	$i^3 = -i$
LASTZ	->	0 + j1	
5, **Z^X**	->	0 + j1	$i^5 = i$

(Note in this example that for enhanced usability **Z^X** stores the original argument in the LastZ register, even though it wasn't strictly located in the **Z** level of the complex stack. The same behavior is implemented in **X^Z**.)

Alternatively, using **W^Z** and **ZREPL**:

1, ENTER^, 0, **ZREPL**	->	0 + j1	i
0, ENTER^, 2, **W^Z**	->	-1 + j0	$i^2 = -1$
ZRND	->	0 + j1	i
0, ENTER^, 3, **W^Z**	->	0 - j1	$i^3 = -i$
ZRND	->	0 + j1	i
0, ENTER^, 4, **W^Z**	->	1 + j0	$i^4 = 1$
ZRND	->	0 + j1	i
0, ENTER^, 5, **W^Z**	->	0 + j1	$i^5 = i$

Examples.- Calculate the value of z = 2^1/(1+i), and z=(1+i)^1/2

These two have a very similar key sequence, but they have different meaning:

Solution: 1, ENTER^, ENTER^, 2, **X^1/Z** -> 1,330 – J0,480
Solution: 1, ENTER^, ENTER^, 2, **Z^1/X** -> 1,099 + j0,455

NXTNRT	**Next value of Nth. Root**	Z=z0*e^j 2π/N	z0 is the principal value

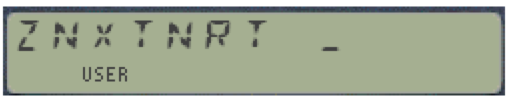
In its general form, the solution to the Nth. root in the complex plane admits multiple solutions. This is because of its logarithmic nature, since the logarithm is a multi-valued function (see discussion in next section).

Z^1/N = e^[Ln(z)/N] = e^[Ln(|z|)+i(α+2π)]/N = e^[Ln(|z|)+iα]/N * e^j 2π/N

From this we derive the general expression**:** **Next(z^1/N) = z^1/N * e^(j 2 π /N)**

thus there are N different Nth. roots, all separated by (2π over N). See the geometric interpretation on section 7 ahead for further discussion on this.

When executed in a program or RUN mode, data entry for this function expects N in the X register, and *z* in the **Z** complex stack level. However when the Complex Keyboard shortcut is used, *the index N is prompted as part of the entry sequence* – a much more convenient way.



Shortcut: Z, Z, SHIFT, SQRT

Example:- Calculate the *two* square roots of 1.

0, ENTER^, 1, **ZENTER^,** 2, **Z^1/X** -> 1+j0
2, **NXTNRT** (plus **ZRND**) -> -1+j0

Note that the previous root is temporarily stored in the LastZ register:
LASTZ -> 1+j0 (previous root)

See section 9 for a general application program to calculate the n different Nth. Roots of a complex number

Example.- Calculate the *three* cubic roots of 8.

Using "direct" data entering: [Im(z), ENTER^, Re(z)]

0, ENTER^, 8, **ZENTER^**, 3, **Z^1/X** -> 2+j0
3, **NXTNRT** -> -1,000+j1,732
3, **NXTNRT** -> -1,000-j1,732

Note: for this example use the *Complex Keyboard* **ZKBRD** to execute **NXTNRT**, as follows:

Z, **Z**, SHIFT, SQRT, and then input 3 at the last prompt.

Example: Calculate both quadratic roots of 1+2i.

2, ENTER^, 1, **ZSQRT** gives the first root: z= 1,272+0,786 j
2, **NXTNRT** gives the second root: z=-1,272-0,786 j
2, **NXTNRT** reverts to the first, principal value, of the root.

This verifies that both roots are in fact on the same straight line, separated 180 degrees from each other and with the same module.

Example: Calculate the three cubic roots of 1+2i.

2, ENTER^, 1, **ZENTER^** inputs z in the complex stack level **Z**
3, 1/X, **Z^X** gives the main root: z= 1,220+0,472 j
3, **NXTNRT** gives the second root: z=-1,018+0,82 j
3, **NXTNRT** give the third and last: z=-0,201-1,292 j

In the next section we'll discuss the logarithm in the complex plane, a very insightful and indeed interesting case study of the multi-valued functions.

5.3. Complex Logarithm.

Table-x: Logarithm group.

Index	Function	Formula	Description
1	**ZLN**	$Z=\text{Ln}\|z\|+i\alpha$	Principal value of natural logarithm
2	**ZLOG**	Z=Ln(z)/Ln10	Principal value of decimal logarithm
3	**ZWLOG**	Z=Ln(z)/Ln(w)	Base-w logarithm of z
4	**NXTLN**	$Z=z+2\pi$ j	Next value of natural logarithm

The first thing to say is that a rigorous definition of the logarithm in the complex plane requires that its domain be restricted, for if we defined it valid in all the plane, such function wouldn't be continuous, and thus neither *holomorfic* (or expressible as series of power functions).

This can be seen intuitively if we consider that:

Since: $z = |z|*e\^{}i\alpha$, then: $\text{Ln } z = \text{Ln } |z| + \text{Ln } (e\^{}i\alpha) = \text{Ln}(|z|) + i\alpha$

But since $z = |z|*e\^{}i\ (\alpha+2\pi) = |z|*e\^{}i\ (\alpha+4\pi)= .... = |z|*e\^{}i\ (\alpha+2\pi\ n)$

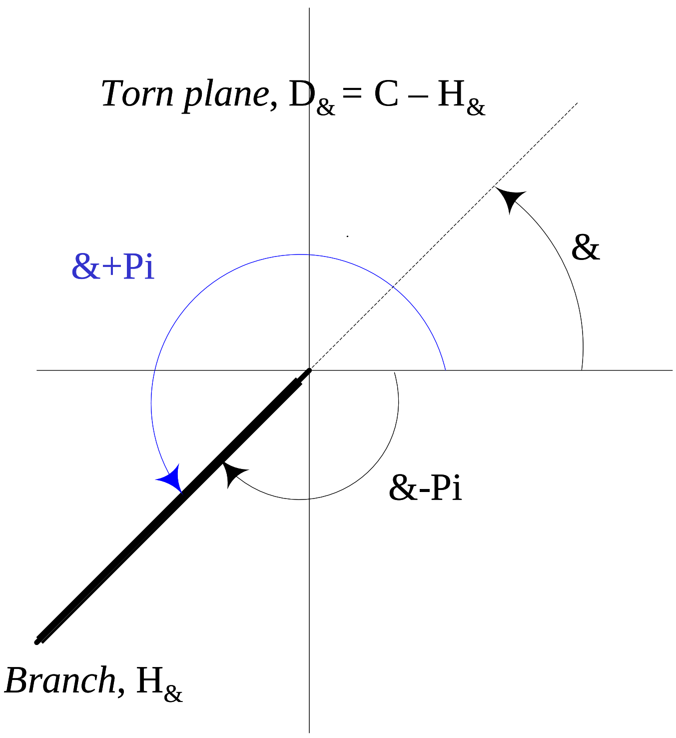
Then we'd equally have multiple values of its logarithm, as follows:

$\text{Ln}(z) = \text{Ln}(|z|) + i\alpha = \text{Ln}(|z|)+i\ (\alpha+2\pi n) = ...$

Or, in general:

$\text{LN } z = \text{Ln}(|z|)+i\ (\alpha+2\pi\ n)$, where n is a natural number.

To deal with this multi-valued nature of the function, mathematicians define the different ***branches of the complex*** *logarithm, (logγ)* as the single one and only logarithm which argument is comprised between $(\gamma-\pi)$ and $(\gamma+\pi)$, thus within the open interval $](\gamma-\pi), (\gamma+\pi)[$



Its domain isn't the whole complex plane, but it excludes a semi-straight line, centered at the origin, that forms an angle & with the real axis, as shown in the picture. Such set is called the "***torn" or cut complex plane** at angle* γ*".* Thus the principal value of the logarithm really should be called Log_0, as it tears (or cuts) the complex plane by the real negative semi-axis, or otherwise γ =0. This means it is *NOT defined* for any negative numbers, and when those need to be subject of its application, a different cut should be chosen.

Therefore all arguments should be comprised between 180 and –180 degrees, as it would correspond to this definition of "Log_0".

In practicality, the values calculated by **ZLN** always lie within this interval,

since they use the internal routines of the calculator, [TOPOL] and [TOREC].

The angle γ should not be confused with the base of the logarithm, which is always the number e –that is, there are natural logarithms.

(See http://en.wikipedia.org/wiki/Branch_point for a more rigorous description of this subject).

After this theoretical discussion, let's see the functions from the 41Z module:-

ZLN	**Natural logarithm**	Z=Ln|z|+iα	Does LastZ

Calculates the principal value of the natural logarithm, using the expression:

Ln z = Ln|z| + iα,   where α = Arg(z)

Example: check that: z=Ln(e^z), for z=(1+i) and z=(2+4i)

1, ENTER^, **ZEXP, ZLN** -> 1,000+j1,000
4, ENTER^, 2, **ZEXP, ZLN** -> 2-j2,283

How do you explain the last result? Is it correct? Try executing **NXTLN** (see below) on it...

NXTLN -> 2+j4,000 - that's more like it!

ZLOG	**Decimal logarithm**	Z=Ln(z)/Ln10	Does LastZ

Calculates the principal value of the decimal logarithm using the expression:

Log z = Ln z / Ln(10)

Example: check that: z=Log(10^z), for z=(1+i) and z=(2+4i)

1, ENTER^, **ZALOG, ZLOG** -> 1(1+j)
4, ENTER^, 2, **ZALOG, ZLOG** -> 2+j1,271

How do you explain the last result? Is it correct? Have you found a bug on the 41Z?

ZWLOG	**Base-W Logarithm**	Z=Ln(z)/Ln(w)	Does LastZ

General case of ZLOG, which has w=10. This is a dual function,

Log z = Ln z / Ln w

NXTLN	**Next Natural logarithm**	Z=z0+2π j	z0 is the principal value

Calculates the next value of the natural logarithm, using the expression:

Next(Ln z) = Ln(z) + 2π j

So the different logarithms are "separated" 2π in their imaginary parts. This works both "going up" as well as "going down", thus each time **NXTLN** is executed two values are calculated and placed in complex levels Z and W. You can use **Z<>W** to see them both.

6. Complex geometry.

The next set of functions admits a geometrical interpretation for their results. Perhaps one of the earliest ways to approach the complex numbers was with the analogy where the real and imaginary parts are equivalent to the two coordinates in a geometric plane.

Table-6.1: Complex geometric group.

Index	Function	Formula	Description
1	**ZMOD**	|z|=SQR(x^2+y^2)	Module or magnitude of a complex number
2	**ZARG**	α =ATAN(y/x)	Phase or angle of a complex number
3a	**ZNEG**	Z=-z	Opposite of a complex number
3b	**ZCHSX**	Z=(-1)^x * z	Opposite (by X) of a complex number
4	**ZCONJ**	Z=x-y j	Conjugated of a complex number
5	**ZSIGN**	Z=z/|z|	Sign of a complex number
6	**ZNORM**	Z=|z|^2	Norm of a complex number
7	**Z*I**	Z=z*i	Rotates z 90 degrees counter clockwise
8	**Z/I**	Z=z/i	Rotates z 90 degrees clockwise

In fact, various complex operations admit a geometrical interpretation. An excellent reference source for this can be found at the following URL: http://www.clarku.edu/~djoyce/complex.

Let's see the functions in detail.

ZMOD	**Module of z**	|z|=SQR(x^2+y^2)	Does LastZ
ZARG	**Argument of z**	α=ATAN(y/x)	Does LastZ

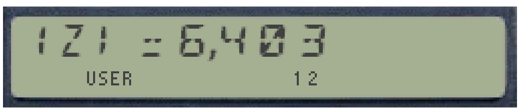
This pair of functions calculates the module (or magnitude) and the argument (or angle) of a complex number, given by the well-known expressions:

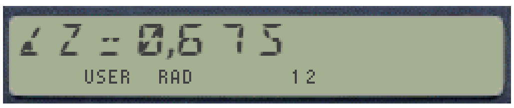
$$|z| = SQR(x^2 + y^2)$$
$$\alpha = ATAN(y/x)$$

Since they use the internal [TOPOL] routine (like R-P does), the argument will always be given between 180 and –180 degrees (or equivalent in the selected angular mode).

The result is saved in the complex **Z** register, and the real X,Y stack levels – as a complex number with zero imaginary part. The original complex number is stored in the Last**Z** register. The other complex stack levels **W, V, U** aren't disturbed.

These functions display a meaningful description when used in run mode, as can be seen in the pictures below, for z= 5+4 j and RAD mode.



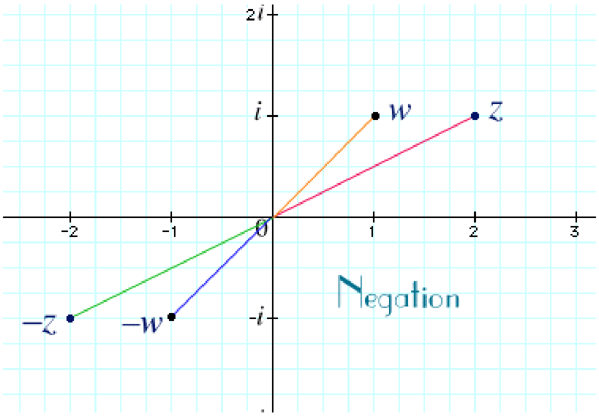


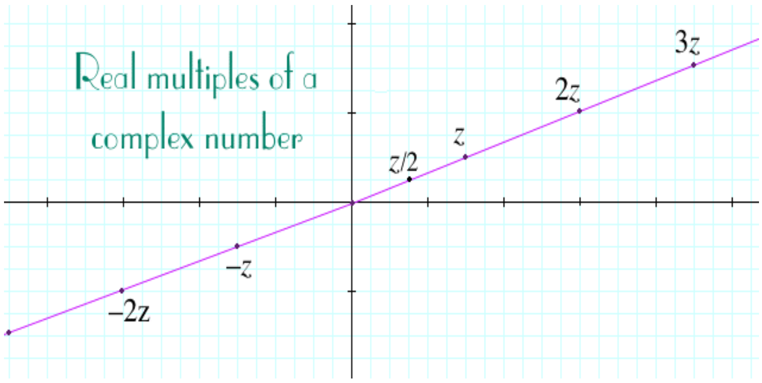
ZNEG	**Opposite of z**	Z=-z	Does LastZ
ZCHSX	**Opposite of z by X**	Z=(-1)^x * z	Does LastZ
ZCONJ	**Conjugate of z**	Z=x-y j	Does LastZ

This pair of functions calculate the opposite- or the multiple-opposite by (-1)^x - and the conjugate of a complex number z=x+y i, as follows:

-z = -x –y I,   and z^* = x – y I

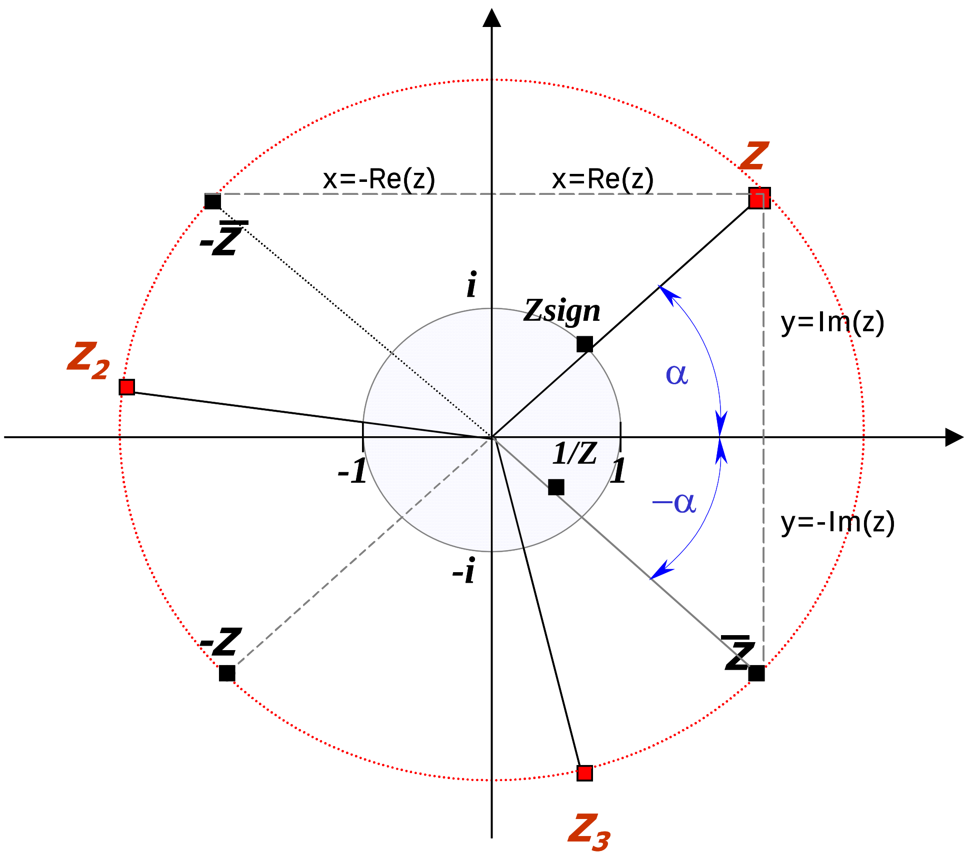
See the figure below for the geometric interpretation of **ZNEG** and multiplication by real numbers:





<table>
<tr><td>ZSIGN</td><td>Module of z</td><td>Z=z/|z|</td><td>Does LastZ</td></tr>
</table>

This function calculates the sign of a complex number. As an extension to the SIGN function for the real domain, it is a complex number with magnitude of one (i.e. located on the unit circle), that indicates the direction of the given original number. Thus obviously: Zsign = z / |z|



The figure above shows the unit circle and the relative position in the complex plane for the opposite (-z), conjugate (zc), and opposite conjugate (-zc) of a given number z.

Note that the inverse of z (1/z) will be located inside of the unit circle, and over the direction defined by the negative of its argument [-Arg(z)]

Note that if z happens to be a cubic root of another number (i.e. z^3), then the other two roots (z_2 and z_3) will have the same module and be located at 120 degrees from each other, on the red circle line.

ZNORM	Norm of z	‖Z‖=|z|^2	Does LastZ

This function calculates the norm of a complex number, also known as the square of its module"

$||z|| = |z|^2$; thus: $Znorm = x^2 + y^2$

When executed in run mode, the display shows a meaningful representation for it, like in the example below, also for z = 4 + 5 j :

Z*I	Multiply by i	Z=z*i	Rotates z 90 deg ccw
Z/I	Divide by i	Z=z/i	Rotates z 90 deg cw

The main role of these two functions is as subroutines for the trigonometric set, and they are also provided for completion sake. Their geometric interpretation is a 90 degrees rotation of the complex number either clockwise or counter-clockwise respectively.

6.2 Complex Comparisons.

The 41Z module includes a comprehensive set of comparison checks, based on the complex numbers themselves and their modules (for relative position in the complex plane). Checks for purely real or imaginary cases are also provided. The main utilization for these functions is in program mode, as conditional decisions under program control based on the different values.

Table 6.2. Complex comparisons function group.

Index	Function	Formula	Description
1	**Z=0?**	Is z=0?	Checks if z is zero
2	**Z#0?**	Is z#0?	Checks if z is not zero
3	**Z=I?**	Is z=i?	Checks if z is the imaginary unit
4	**Z=W?**	Is z=w?	Checks if z and w are the same
5	**Z=WR?**	Is z=w rounded?	Checks if rounded z and rounded w are the same
6	**Z#W?**	Is z#w?	Checks if z and w are different
7	**ZUNIT?**	Is |z|=1?	Checks if z is on the unit circle
8	**ZIN?**	Is |z|<1?	Checks whether z is inside the unit circle
9	**ZOUT?**	Is |z|>1?	Checks whether z is outside the unit circle
10	**ZREAL?**	Is z a real number?	Checks whether Im(z)=0
11	**ZIMAG?**	Is z true imaginary?	Checks whether Re(z)=0
12	**ZINT?**	Is z true integer?	Checks whether Im(z)=0 and FRC[Re(z)]=0

It's well know that, contrary to real numbers, the complex plane isn't an ordered domain. Thus we can't establish ordered relationships between two complex numbers like they are done with real ones (like x>y, x<y?, etc.).

There are however a few important cases that can also be used with complex numbers, as defined by the following functions.- As it is standard, they respond to the "*do if true*" logic, skipping the next program line when false.

Z=W?	**Compares z with w**	Are they equal?	
Z#W?	**Compares z with w**	Are they different?	
Z=WR?	**Compares z with w rounded**	Are they equal?	
Z=0?	**Compares z with zero**	Are they equal?	
Z#0?	**Compares z with zero**	Are they different?	
Z=I?	**Compares z with i**	Are they equal?	

The first two functions compare the contents of the **Z** and **W** stack levels, checking for equal values of both the real and imaginary parts.

z=w iff Re(z)=Re(w) and Im(z)=Im(w)

The third function, **Z=WR?** will establish the comparison *on the rounded values of the four real numbers*, according to the current display settings on the calculator (i.e. number of decimal places shown). This is useful when programming iterative calculations involving conditional decisions.

Rnd(z) = Rnd(w) iff rnd[Re(z)]=rnd[Re(w)] and rnd(Im(z)] = rnd[Im(w)]

The remaining three functions on the table are particular applications of the general cases, checking whether the **Z** complex stack level contains zero or the imaginary unit:

z=0 iff Re(z)=0 and Im(z)=0
z=i iff Re(z)=0 and Im(z)=1

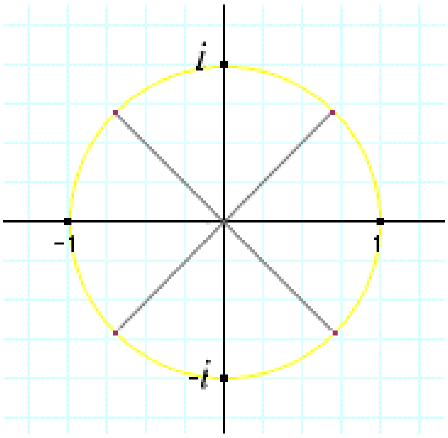
Some of the inverse comparisons can be made by using standard functions, as follows:

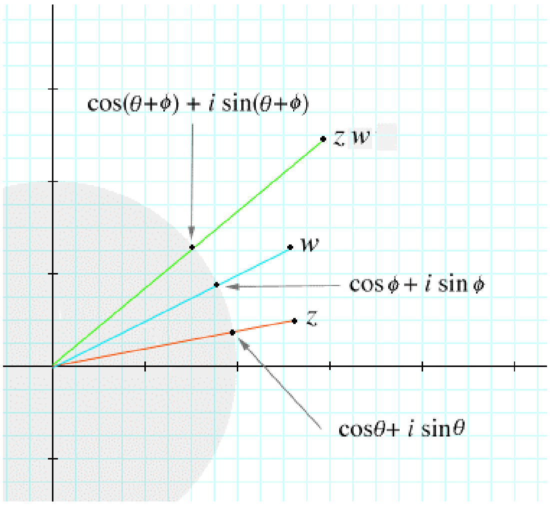
- use **X#0?** to check for Z#0? condition
- Use **X#0?** to check for Z#I? Condition

ZUNIT?	**Checks if z is on the unit circle**		
ZIN?	**Checks if |z|<1**		
ZOUT?	**Checks if |z|>1**		

These three functions base the comparison on the actual location of the complex number referred to the unit circle: inside of it, on it, or outside of it. The comparison is done using the number's modulus, as a measure of the distance between the number and the origin.

Unit Circle

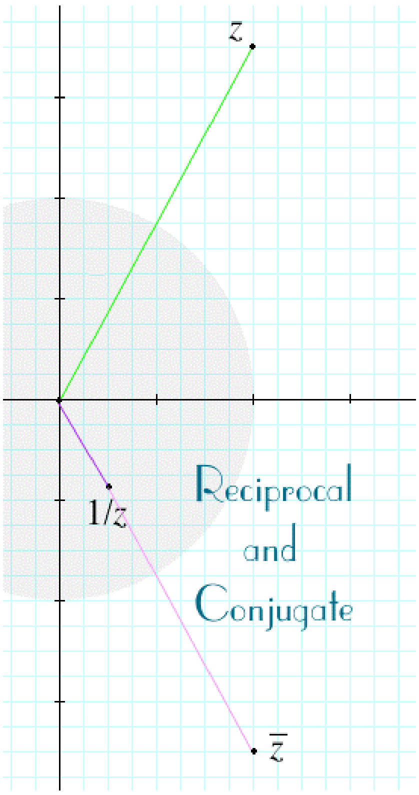


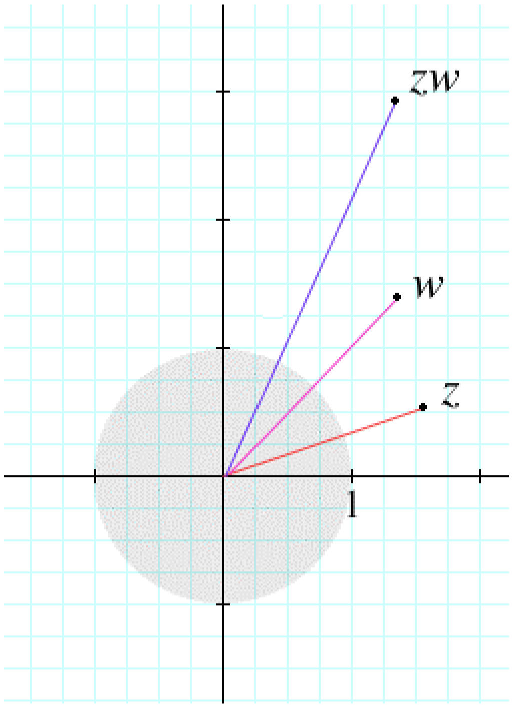


Example: For z=4+5j , calculate its sign and verify that it's located on the unit circle:

5, ENTER^, 4, **ZSIGN**,	→ result: ZSign = 0,62+0,78 j
ZUNIT?	→ result: "YES"
DEG**, POLAR**	→ result: 1,00 < 51,34 (in degrees)

In program mode the behavior is ruled by the "do if true" rule, skipping the next line if false.





ZREAL?	**Checks if z is purely real**		
ZIMAG?	**Checks if z is purely imaginary**		
ZINT?	**Checks if z is an integer**		

The first two functions check whether the complex number is purely a real or imaginary number.

Do not mistake these comparison functions with the other pair, {**ZREAL** and **ZIMAG**}, which cause the number to change to become either real or imaginary – nor with {**ZREAL^** and **ZIMAG^**}, which are used to input complex numbers of the selected type based on the value stored in the real stack level X.

The third one extends the scope of ZREAL?, adding the condition of being a true integer number:

- **ZINT?** True means **ZREAL?** True, and FRC(Re(z))=0

Do not mistake it with **ZINT**, which causes the complex number to have no decimal figures in BOTH its real and imaginary parts – *therefore it's result not a Real number*!

ZINT? is used in the FOCAL programs to calculate Bessel Function, as a quick an effective way to determine if the order is integer – which triggers different expressions for the formulas.

Like it occurs with any built-in comparison function, there's no action taken on the original number, which will remain unchanged.

7. Complex Trigonometry.

Table 7.1. Complex trigonometry function group.

Index	Function	Formula	Description
1	**ZSIN**	sin z = -i *sinh (iz)	Complex Sine
2	**ZCOS**	cos z = cosh (iz)	Complex Cosine
3	**ZTAN**	tan z = - i * tanh (iz)	Complex Tangent
4	**ZHSIN**	sinh z = 1/2 * [e^z - e^-z]	Complex Hyperbolic Sine
5	**ZHCOS**	cosh z = 1/2 * [e^z + e^-z]	Complex Hyperbolic Cosine
6	**ZHTAN**	tanh z = (e^z-e^-z)/(e^z+e^-z)	Complex Hyperbolic Tangent

And their inverses:

7	**ZASIN**	asin z = -i * asinh (iz)	Complex Inverse Sine
8	**ZACOS**	acos z = π/2 - asin z	Complex inverse Cosine
9	**ZATAN**	atan z = -i * atanh (iz)	Complex Inverse Tangent
10	**ZHASIN**	asinh z = Ln[z + SQ(z^2 + 1)]	Complex Inverse Hyperbolic Sine
11	**ZHACOS**	acosh z = Ln[z + SQ(z^2 - 1)]	Complex Inverse Hyperbolic Cosine
12	**ZHATAN**	atanh z = 1/2 * Ln[(1+z)/(1-z)]	Complex Inverse Hyperbolic Tangent

This section covers all the trigonometric and hyperbolic functions, providing the 41Z with a complete function set. In fact, their formulas would suggest that despite their distinct grouping, they are nothing more than particular examples of logarithm and exponential functions (kind of *"logarithms in disguise"*).

Their usage is simple: the argument is taken from the complex-**Z** level and *always* saved on the LastZ register. The result is placed on the complex-**Z** level. Levels **W, V, U** are preserved in all cases, including the more involved calculations with **ZTAN** and **ZATAN** (those with the devilish names), for which extensive use of scratch and temporary internal registers is made.

The formulas used in the 41Z are:

sin z = -i *sinh (iz)
cos z = cosh (iz)
tan z = - i * tanh (iz)

asin z = -i * asinh (iz)
acos z = π /2 – asin z
atan z = -i * atanh (iz)

sinh z = 1/2 * $[e^z - e^{-z}]$
cosh z = 1/2 * $[e^z + e^{-z}]$
tanh z = $(e^z - e^{-z})/(e^z + e^{-z})$

asinh z = Ln[z + SQ(z^2 + 1)]
acosh z = Ln[z + SQ(z^2 - 1)]
atanh z = 1/2 * Ln[(1+z)/(1-z)]

So we see that interestingly enough, the hyperbolic functions are used as the primary ones, also when the standard trigonometric functions are required. This could have also been done the other way around, with no particular reason why the actual implementation was chosen.

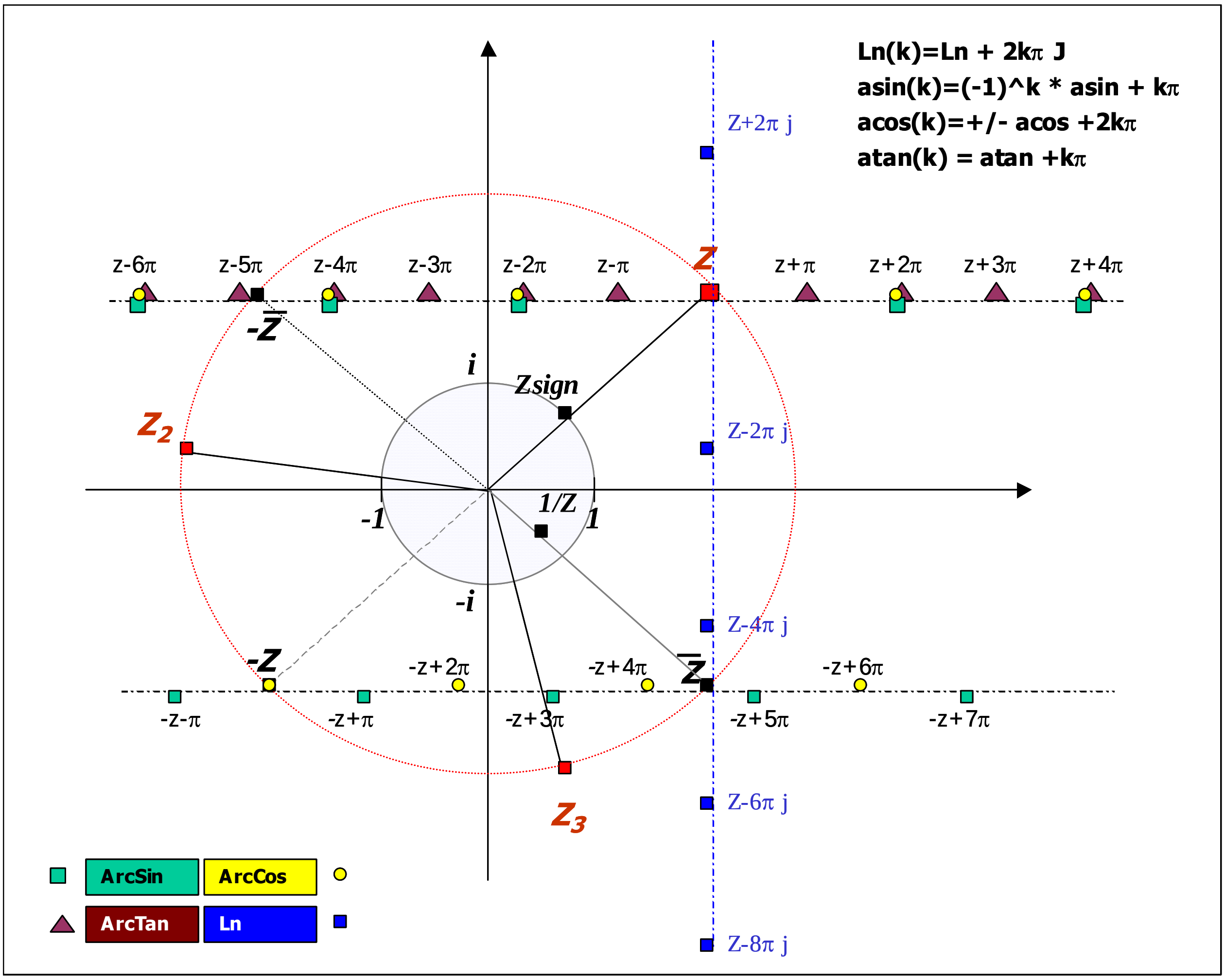
Example. Because of their logarithmic nature, also the inverse trigonometric and hyperbolic functions will be multi-valued. Write a routine to calculate all the multiple values of ASIN z.

01 *LBL "ZASIN"*
02 ZASIN
03 ZSTO
04 ZAVIEW
05 E
06 STO 02
07 LBL 00
08 ZRCL
09 ZNEG
10 ZSTO
11 RCL 02
12 PI
13 *
14 +
15 ZAVIEW
16 PSE
17 E
18 ST+ 02
19 GTO 00
20 END

The 41Z module now also includes new functions to calculate next values for complex ASIN, ACOS and ATAN, as follows: **NXTASN, NXTACS**, and **NXTATN**. Using the first one the program above changes to this very simplified way:

```
01  LBL "ZASIN2"      04  ZAVIEW      07  END
02  ZASIN             05  NXTASN
03  LBL 00            06  GTO 00
```

Using the general expressions we can obtain the multiple values of a given function from its principal value "**Z**" of a given function, as follows:



- the multiple values for ASIN(z) -in green squares- are placed on the two straight lines parallel to the x axis, y=Im[ASIN(z)] and y=–Im[ASIN(z)], and are separated at intervals of 2π length on each line.

- the multiple values for ACOS(z) –in yellow circles– are placed on the same two straight lines, and are separated at intervals of 2π length on each line.

- the multiple values for ATAN(z) –in brown triangles- are placed on the upper of those straight lines, separated at intervals of π length on it.

- the multiple values for Ln(z) –in blue squares- are placed on the vertical straight line x=Re[LN(z)], and separated at intervals of 2π length on it.

- the three different values for z^1/3 are placed in the circle r=|z|^1/3, and are separated at 120 degrees from each other (angular interval).

NXTASN	**Next Complex ASIN**		Does LastZ
NXTACS	**Next Complex ACOS**		Does LastZ
NXTATN	**Next Complex ATAN**		Does LastZ

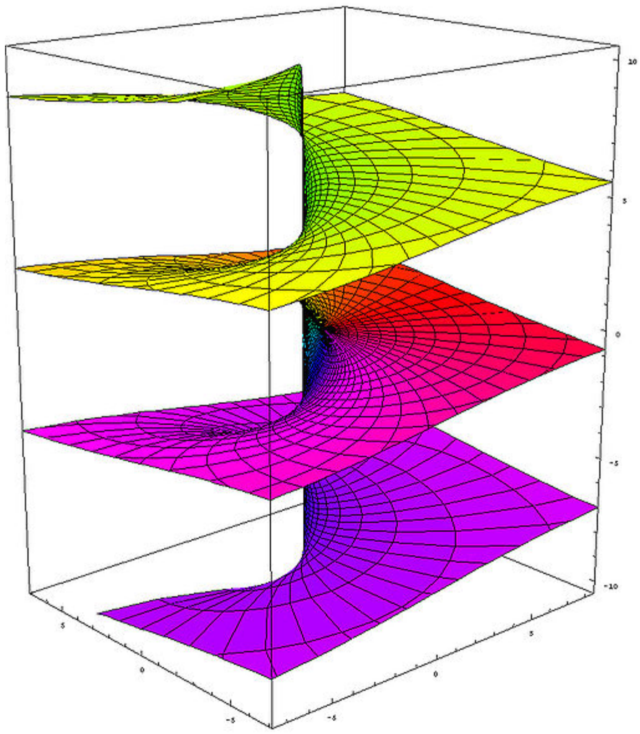
Let z0 be the principal value of the corresponding inverse trigonometric function. Each of these three functions returns *two* values, z1 and z1' placed in complex stack levels **Z** and **W**. z1 will be shown if the function is executed in RUN mode. You can use **Z<>W** to see the value stored in **W** (that is, z1')

The NEXT values z and z1' are and given by the following recursion formulas:

Next ZASIN:
Z1 = Z0 + 2 pi
Z1'= -Z0 + pi

Next ZACOS:
Z1 = Z0+ 2 pi
Z1' = -Z0 + 2 pi

Next ZATAN:
Z1=Z0 + pi
Z1'= Z0 – pi



The figure on the right plots the multi-valued imaginary part of the complex logarithm function, which shows the branches. As a complex number *z* goes around the origin, the imaginary part of the logarithm goes up or down:

For further information on multi-valued complex functions see the following excellent reference: http://en.wikipedia.org/wiki/Branch_point

See section 9 ahead for further details on multi-valued functions, with the FOCAL driver program **ZMTV** (ZMulTiValue) that calculates all the consecutive results of the eight multi-value functions.

8. 2D-vectors or complex numbers?

One of the common applications for complex numbers is their treatment as 2D vectors. This section covers the functions in 41Z that deal with vector operations between 2 complex numbers.

Table 8.1. 2D vectors function group.

Index	Function	Formula	Description
1	**ZWANG**	Arg(ZW) = Arg(Z) - Arg(W)	Angle between 2 vectors
2	**ZWDIST**	|W-Z| = SQR[(Wx-Zx)^2 - (Wy-Zy)^2]	Distance between 2 points
3	**ZWDOT**	Z*W = Zx*Wx + Zy*Wy	2D vector Dot product
4	**ZWCROSS**	Z x W = |z| *|w| *Sin(Angle)	2D vector Cross product
5	**ZWDET**	|ZW| = Wx*Zy - Wy*Zx	2D determinant
6	**ZWLINE**	a=(Y1-Y2) / (X1-X2) b=Y2 - a*X2	Equation of line through two points

These functions use **W** and **Z** levels of the complex stack, leaving the result in level **Z** after performing complex stack drop. The original contents of **Z** is saved in the LastZ register. The following screen captures from V41 show the different displays for these functions:

Let z = 4 <45 degrees, and w= 3 <75 degrees .

45, ENTER^, 4, **ZREC** -> 2,828(1+j)
ZREPL [don't forget or Z will be overwritten]
75, ENTER^, 3, **ZREC** -> 0,776 + 2,898J

1. ZWANG,- angle defined between both vectors (in degrees in this case)
2. ZRDN , LASTZ, **ZWDIST** – distance between both complex numbers

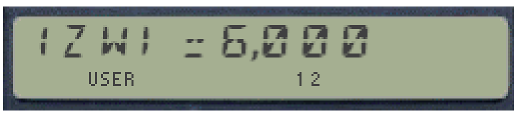
and


The angle will be expressed in the selected angular unit.

3. ZRDN , LASTZ, **ZWDOT** - dot product of both vectors
4. ZRDN, LASTZ, **ZWCROSS** - magnitude of the cross product of both vectors

and


5. ZRDN, LASTZ, **ZWDET** - magnitude of the determinant of both vectors
6. ZRDN, LASTZ, **ZWLINE** - equation of the straight line linking both points



and


(*) Note that despite having a simpler formula, ZWDET shows less precision than ZWCROSS.

9. It's a Gamma/Zeta world out there.

This section describes the different functions and programs included on the 41Z that deal with the calculation of the Gamma and Zeta functions in the complex plane. A group of five functions in total, two completely written in machine code and three as FOCAL programs, plus a couple of example application programs to complement it.

Table 9.1. Gamma function group.

ZGAMMA	**Complex Gamma function**	for z#-k, k=integer	Does LastZ
ZGPRD	**Auxiliary Product**	PROD[(z+n); n=1,..6]	Does LastZ
ZPSI	**Complex Digamma (Psi)**	see below	FOCAL program
ZLNG	**Gamma Logarithm**	see below	FOCAL program
ZZETA	**Complex Riemann Zeta**	For z#1	FOCAL program

ZGAMMA uses the Lanczos approximation to compute the value of Gamma. An excellent reference source is found under http://www.rskey.org/gamma.htm, written by Viktor T. Toth. Remark that ZGAMMA is implemented completely in machine code, even for Re(z)<0 using the reflection formula for analytical continuation.

For complex numbers on the positive semi-plane [Re(z)>0], the formula used is as follows

$$\Gamma(z) = \frac{\sum_{n=0..N} q_n z^n}{\prod_{n=0..N}(z+n)}(z+5.5)^{z+0.5}e^{-(z+5.5)}$$

$q_0 =$	75122.6331530
$q_1 =$	80916.6278952
$q_2 =$	36308.2951477
$q_3 =$	8687.24529705
$q_4 =$	1168.92649479
$q_5 =$	83.8676043424
$q_6 =$	2.5066282

And the following identity (reflection formula) is used for numbers in the negative semi-plane: [Re(z)<0]: which can be re-written as: **G(z) * G(-z) = -π / [z*Sin(π z)]**

$$\Gamma(1-z)\,\Gamma(z) = \frac{\pi}{\sin(\pi z)}$$

Example: Calculate G(1+i)

1, ENTER^, **ZGAMMA** -> "RUNNING...", followed by -> 0,498-j0,155

Example: Verify that G(1/2) = SQR(π)

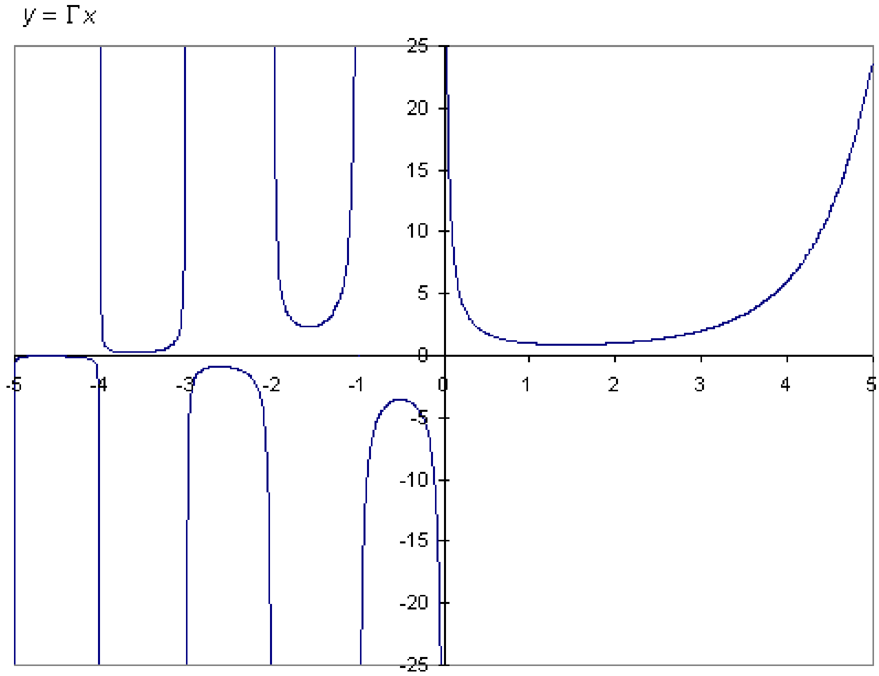
0, ENTER^, 0.5, **ZGAMMA** -> 1,772 + j0
PI, SQRT, **ZREAL^**, **Z-** -> -2,00E-9 + j0

Example: Calculate G(-1.5+i)

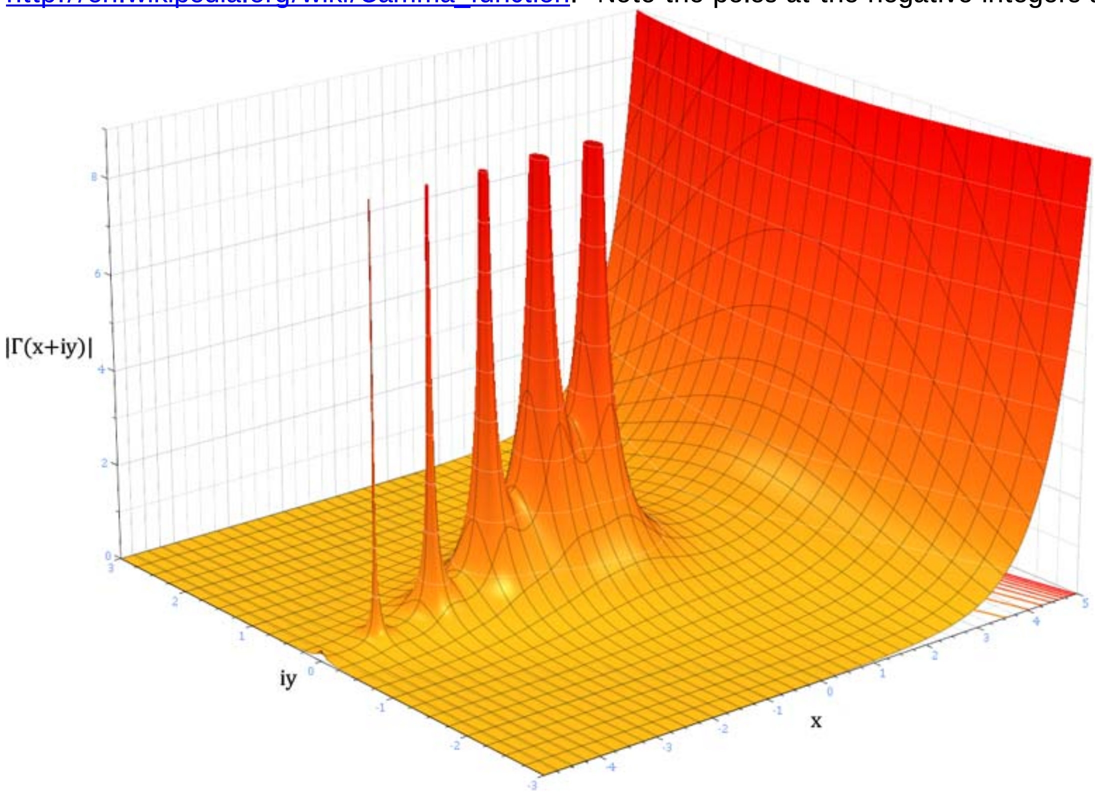
1, ENTER^, 1.5, CHS, **ZGAMMA** -> 0,191 + j0,174

For cases when the real part of the argument is negative [Re(z)<0], **ZGAMMA** uses a FOCAL program to compute the reflection formula – all internal and transparent to the user.

The graphic below (also from the same web site) shows Gamma for real arguments. Notice the poles at x=0 and negative integers.



The following graphic showing the ***module* of the Complex Gamma** function is taken from http://en.wikipedia.org/wiki/Gamma_function.- Note the poles at the negative integers and zero.



Example: Use **ZLNG** to calculate G(1+i) and compare it with the value obtained by **ZGAMMA**

1, ENTER^, **ZGAMMA, LASTZ, ZLNG, ZEXP, Z-** -> 2,400E-9+j3,000E-10

Program listings.-

The two FOCAL programs listed below calculate the Digamma and the Gamma functions for complex arguments. The first one is an example using the asymptotic approximation as described below, whilst the second one is an extension of the MCODE function **ZGAMMA**, using the reflection formula for arguments with Re(z)<1 (programmed in turn as another MCODE function, **ZGNZG**).

01	*LBL "ZPSI"*	26	Z/	*01*	*LBL "ZG"*
02	ZREPL	27	21	02	ZENTER^
03	7 E-3	28	1/X	03	X<>Y
04	STO O	29	ZREAL^	04	X#0?
05	CLZ	30	Z-	05	*GTO 00*
06	*LBL 00*	31	Z*	06	X<>Y
07	Z<>W	32	0,1	07	X>0?
08	RCL O	33	+	08	*GTO 00*
09	INT	34	Z*	09	INT
10	+	35	1	10	LASTX
11	ZINV	36	-	11	X#Y?
12	Z+	37	Z*	12	*GTO 00*
13	ISG O	38	12	13	0
14	*GTO 00*	39	ZREAL^	14	1/X
15	ZSTO	40	Z/	15	*LBL 00*
16	1	41	ZRCL (00)	16	ZRDN
17	Z<>W	42	ZLN	17	CF 00
18	8	43	LASTZ	18	X<0?
19	+	44	ZHALF	19	SF 00
20	ZINV	45	Z+	20	FS? 00
21	ZSTO (00)	46	Z-	21	ZNEG
22	Z^2	47	ZRCL	22	*ZGAMMA*
23	ZREPL	48	1	23	FC? 00
24	20	49	Z-	24	*GTO 01*
25	ZREAL^	50	ZAVIEW	25	LASTZ
		51	END	26	*ZGNZG*
				27	Z<>W
for x>8				28	Z/
Psi(x) = ln x - 1/(2x) -1/(12x^2) + 1/(120x^4) - 1/(252x^6) + 1/(240x^8)				29	*LBL 01*
together with the relationship: Psi(x+1) = Psi(x) + 1/x				30	ZAVIEW
				31	END

Approximation for Digamma when x>8

$$\Psi(x) = \log(x) - \frac{1}{2x} - \frac{1}{12x^2} + \frac{1}{120x^4} - \frac{1}{252x^6} + O\left(\frac{1}{x^8}\right)$$

programmed as: **u^2{[(u^2/20-1/21)u^2 + 1/10]u^2 –1}/12 – [Ln u + u/2]**,

where **u=1/x**; and using the following precision correction factor:

$$\Psi(x+1) = \Psi(x) + \frac{1}{x}.$$

The next expression shows Stirling's approximation for Gamma:

$$\Gamma(z) \approx \sqrt{\frac{2\pi}{z}}\left(\frac{z}{e}\sqrt{z\sinh\frac{1}{z} + \frac{1}{810z^6}}\right)^z,$$

The following two programs calculate the Logarithm of the Gamma function for complex arguments. The first one uses the Stirling approximation, with a *correction factor* to increase the precision of the calculation. This takes advantage of the **ZGPRD** function, also used in the Lanczos approximation.

$$2\ln\Gamma(z) \approx \ln(2\pi) - \ln z + z\left(2\ln z + \ln\left(z\sinh\frac{1}{z} + \frac{1}{810z^6}\right) - 2\right),$$

correction factor: **LnG(z) = LnG(z+7) - Ln[PROD(z+k)|k=1,2..6]**

The second one applies the direct definition by calculating the summation until there's no additional contribution to the partial result when adding more terms. In addition to being much slower than the Stirling method, this is also dependent of the display precision settings and thus not the recommended approach. It is not included on the 41Z but nevertheless is an interesting example of the utilization of some of its functions, like **Z=WR?** and the memory storage registers, **ZSTO** and **ZRCL**.

$$\ln\Gamma(z) = -\gamma z - \ln z + \sum_{k=1}^{\infty}\left[\frac{z}{k} - \ln\left(1+\frac{z}{k}\right)\right]$$

01	LBL "ZLNG"		01	LBL "ZLNG2"
02	7		02	1
03	+		03	STO 02
04	ZSTO (00)		04	RDN
05	*Text-0*	*NOP*	05	ZSTO (00)
06	6		06	*XEQ 05*
07	CHS		07	*LBL 00*
08	Z^X		08	ZENTER^
09	810		09	*XEQ 05*
10	ST/ Z		10	Z+
11	/		11	Z=WR?
12	ZRCL (00)		12	*GTO 02*
13	ZINV		13	*GTO 00*
14	ZSINH		14	*LBL 02*
15	ZRCL (00)		15	ZRCL (00)
16	Z*		16	ZLN
17	Z+		17	Z-
18	ZLN		18	ZRCL (00)
19	ZRCL (00)		19	0,5772156649
20	ZLN		20	ST* Z
21	ZDBL		21	*
22	Z+		22	Z-
23	2		23	ZAVIEW
24	-		24	RTN
25	ZRCL (00)		25	*LBL 05*
26	Z*		26	ZRCL (00)
27	ZRCL (00)		27	RCL 02
28	ZLN		28	ST/ Z
29	Z-		29	/
30	PI		30	ZENTER^
31	ST+ X		31	1
32	LN		32	+
33	+		33	ZLN
34	ZHALF		34	Z-
35	ZRCL (00)		35	1
36	*Text-0*	*NOP*	36	ST+ 02
37	7		37	RDN
38	-		38	END
39	*ZGPRD*			
40	ZLN			
41	Z-			
42	ZAVIEW			
43	END			

Max ZREG#	Size
n/a	1
0	2
	3
1	4
	5
2	6
	7
3	8
	9
4	10
	11
5	12
	13
6	14
	15
7	16
	17
8	18
	19
9	20
	21
10	22
	23
11	24
	25
...	...

The table on the right shows the correspondence between the complex register number(CRnn) and the required SIZE in the calculator. Note that a minimum of SIZE 003 is required for CR00 to exist.

10. Application programs.

The following functions in the 41Z are in reality FOCAL programs, included as application examples because of their applicability and as a way to illustrate actual programming of the complex number functions of the module.

Index	Function	Description	Author
1	**ZQRT**	Roots of Quadratic equation	AM
2	**ZCRT**	Roots of Cubic equation	AM
3	**ZMTV**	Multi-valued functions	AM
4	**ZPROOT**	Roots of a polynomial of any degree	Valentín Albillo
5	**ZSOLVE**	Solves f(z)=0 by secant method	AM
6	**ZWL**	Lambert-W function	JM Baillard & AM
7	**ZAWL**	Inverse of Lambert-W	AM

Note: Most of these functions appear on CAT'2 as M-Code entries, instead of as FOCAL programs. This is achieved by using a clever technique shown by William E. Wilder (author of the BLDROM), which allows cleaner and convenient program listings (no ugly "XROM" description before the program title).

These programs can still be copied into main memory using COPY as usual, but won't have the global label. The drawback is that they can't be "looked-up" using GTO + global LBL, since there's no global LBL for them.

10.1 Solution of quadratic and cubic equations.

ZCRT	**Roots of cubic equation**	Main routine	
ZQRT	**Roots of quadratic equation**	Main routine	

ZQRT Solves the roots of a quadratic equation with complex coefficients, as follows:

$C_1 * z^2 + C_2 * z + C_3 = 0$; where C_1, C_2, C_3, and z are complex numbers

By applying the general formula:

$z_{1,2} = [-C_2 +/- SQR(C_2^2 - 4C_1*C_3)] /2*C_1$

Example: find out the roots of **(1+i)*z²** **+ (-1-i)*z + (1-i) = 0**

ZQUAD	"aZ^2+bZ+c=0", followed by:
	"IM^RE a=?
1, ENTER^, R/S	"IM^RE b=?
1, CHS, ENTER^, R/S	"IM^RE c=?
1, CHS, ENTER^, 1, R/S	"RUNNING…" followed by:
	" 1,300+j0,625"
R/S	" -0,300-j0,625"

We can see that both roots are NOT conjugate of each other, as it occurs with real coefficients.

Program listing.-

1	LBL "ZQUAD"	12	PROMPT	23	ZHALF	34	Z+
2	"aZ^2+bZ+c=0"	13	"RUNNING..."	24	ZNEG	35	ZRDN
3	AVIEW	14	AVIEW	25	ZENTER^	36	Z+
4	PSE	15	LBL "ZQRT"	26	ZENTER^	37	ZRUP
5	"IM^RE a=?"	16	ZENTER^	27	Z^2	38	SF 21
6	PROMPT	17	ZRUP	28	ZRUP	39	ZAVIEW
7	ZENTER^	18	Z/	29	Z-	40	Z<>W
8	"IM^RE b=?"	19	LASTZ	30	ZSQRT	41	CF 21
9	PROMPT	20	ZRUP	31	ZENTER^	42	ZAVIEW
10	ZENTER^	21	Z<>W	32	ZNEG	43	END
11	"IM^RE c=?"	22	Z/	33	ZRUP		

Note that **ZQUAD** is just a driver for **ZQRT,** which expects the three complex coefficients stored in levels **V**, **W**, and **Z** of the complex stack. Note also that *no memory registers are used,* and all calculations are performed using exclusively the complex stack. The core of the program is from lines 16 to 37, or just 21 programming steps to resolve both roots.

The 41Z complex function set and complex stack enables the programmer to treat complex calculations as though they used real numbers, not worrying about the real or imaginary parts but working on the complex number as single entity. In fact, exercising some care, you could almost translate one-to-one many FOCAL programs by replacing the standard functions with the equivalent complex ones. That's why it's important that the function set be as complete as possible, and that the same RPN conventions are followed by the complex stack.

10.2 Lambert W function.

ZWL	**Lambert W(z)**		FOCAL program
ZAWL	**Inverse of Lambert-W**	**z* e^z**	Does LastZ

These two functions provide a dedicated way to compute the Lambert-W function and its inverse. The FOCAL program uses an iterative method to compute W(z), using 1+Ln(z) as initial guess for Re(z)>0, and simply 1+i elsewhere.

This program is based on a real-mode version written by JM Baillard, just applying the seamless transposition method provided by the 41Z module. In the vast majority of cases convergence is provided for all complex arguments, with 8-decimal digits accuracy. It uses the **Z=WR?** Function on FIX 8 mode to determine that two consecutive iterations are equal.

Another version using SOLVE is listed in section 10.5.1, with slightly more accurate results , but significantly slower execution.

10.3 Multi-value Functions.

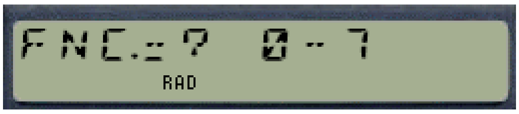
ZMTV	**Multi-valued functions**		

This program calculates all possible values for the multi-valued functions, including the n different $N^{th.}$ roots of a complex number, all the inverse trigonometric and hyperbolic, plus the logarithm itself (source of all the multi-valued scenarios).

Due to the 64-function limit of the 41 ROM FAT structure. these routines are all part of a common entry into the module catalog. The program prompts for a number, from zero to eight where Zero lists the catalog of all possible choices, as follows:

Index	Function	Description	Equivalence
1	**ZACOS**	Inverse Complex Cosine	Same as **NXTACS**
2	**ZACOSH**	Inverse Complex Hyperbolic Cosine	n/a
3	**ZASIN**	Inverse Complex Sine	Same as **NXTASN**
4	**ZASINH**	Inverse Complex Hyperbolic Sine	n/a
5	**ZATAN**	Inverse Complex Tangent	Same as **NXTATN**
6	**ZATANH**	Inverse Complex Hyperbolic Tangent	n/a
7	**ZLN**	Complex Logarithm	Same as **NXTLN**
8	**Z^1/N**	Roots of Complex number	Same as **NXTNRT**

Each case will briefly display the title of the sub-function, and will calculate the principal value followed by all the other values with each subsequent pressing of [R/S].



The first 7 routines expect *z* into the **Z** level of the complex stack. Data entry is the same for all of them except the last one, which expects N in the real-stack register X, and *z* in **Z**. Only the first N values will be different, running into cyclical repetition if continued.

This is a simple program, mostly written to document an example for the 41Z functions. Use it to get familiar with these concepts, and to understand fully the NXT function set as well.

Example: Obtain all values of ASIN [Sin(1+j)]

1, ENTER^, **ZSIN** -> 1,298+j0,635
ZMTV -> "FNC.=? 0-8"
3, R/S -> 1,000+j1
R/S -> 2,142-j1
R/S -> 7,283+j1
R/S -> 8,425-j1
etc...

Alternatively, using the **NXTASN** function:

Note that here we start with the first value of the function, i.e. 1+j

1, ENTER^, **NXTASN** -> 7,238+j1
Z<>W -> 2,142-j1
NXTASN -> 8,425-j1
NXTASN -> 14,708-j1

Program listing.-

Note the use of flag 22 for numeric entry: the catalog of functions will display continuously until one choice is made, (expected between 1 and 8), and all initial prompting will be skipped.

```
LBL "ZMTV"
CF 22
LBL 20
"FCN#.=? 1-8
AVIEW
PSE
PSE
FC? 22
GTO 90
INT
ABS
90
+
RDN
SF 25
GTO IND T
GTO 20
LBL 90
CF 21
"1:- ZACOS"
AVIEW
PSE
"2:- ZACOSH"
AVIEW
PSE
"3:- ZASIN"
AVIEW
PSE
"4:- ZASINH"
AVIEW
PSE
"5:- ZATAN"
AVIEW
PSE
"6:- ZATANH"
AVIEW
PSE
"7:- ZLN"
AVIEW
PSE
"8:- Z^1/N"
AVIEW
PSE
GTO 20
LBL 95
ZATAN
GTO 06
LBL 93
ZASIN
ZSTO
ZAVIEW
E
STO 02
LBL 03
ZRCL
ZNEG
ZSTO
RCL 02
PI
*
+
ZAVIEW
PSE
E
ST+ 02
GTO 03
LBL 91
ZACOS
ZSTO
ZAVIEW
E
STO 02
LBL 01
ZRCL
RCL 02
ST+X
PI
*
STO 03
+
ZAVIEW
PSE
ZRCL
ZNEG
RCL 03
+
ZAVIEW
PSE
E
ST+ 02
GTO 01
LBL 94
ZHASIN
GTO 07
LBL 92
ZHACOS
GTO 07
LBL 96
ZHATAN
LBL 06
ZAVIEW
PSE
PI
+
GTO 06
LBL 97
ZLN
LBL 07
ZAVIEW
PSE
NXTLN
GTO 07
LBL 98
CF 00
"N=?"
PROMPT
ABS
INT
X=0?        zeroth. Root?
RTN
STO 00
E
-           N-1
STO 01
X=0?
SF 00       unit root?
E
+           N
1/X         1/N
Z^X         main value
SF 21
ZAVIEW
FS?C 00
GTO 08
LBL 05
RCL 00
NXTNRT
ZAVIEW
DSE 01
GTO 05
LBL 08
CF 21
END
```

10.4 Roots of Complex Polynomials.

ZPROOT	Roots of Polynomials	By Valentín Albillo	

This program calculates all the roots of a polynomial of degree n, and with complex coefficients. It is therefore *the most general case of polynomial root finders* that can possibly be used, as it also will work when the coefficients are real.

This program is a wonderful example of FOCAL capabilities, and very well showcases the versatility of the HP-41C (even without the 41Z module). It was first published on PPC Technical Notes, PPCTN.

1	LBL "ZPROOT"		44	CF 00	87	E-3	130	GTO 02
2	SIZE?		45	CHS	88	ST+ 01	131	RCL 08
3	"DEGREE=?"		46	STO 04	89	RCL 03	132	ST* Z
4	PROMPT		47	FIX 2	90	STO IND 05	133	*
5	STO Z		48	RND	91	RCL 04	134	DSE 08
6	ST+X	2N	49	FIX 6	92	STO IND 06	135	GTO 02
7	11		50	X#0?	93	DSE 00	136	RTN
8	+	2N+11	51	GTO 01	94	GTO 06	137	LBL 00
9	X>Y?		52	SIGN	95	TONE 5	138	ZENTER^
10	PSIZE		53	STO 04	96	RCL 01	139	RCL 04
11	RCL Z		54	LBL 01	97	INT	140	RCL 03
12	STO 00	N	55	RCL 00	98	E1	141	Z*
13	STO 03	N	56	STO 08	99	-	142	RCL IND 05
14	9,008		57	SF 01	100	E3	143	FS? 01
15	+		58	XEQ 11	101	/	144	RCL 08
16	STO 01	N+9,008	59	R-P	102	ST- 05	145	FS? 01
17	STO 05	N+9,008	60	1/X	103	FIX 3	146	*
18	X<>Y	2N+11	61	STO 07	104	SF 21	147	+
19	E		62	X<>Y	105	LBL 10	148	FS? 00
20	-	2N+10	63	CHS	106	ISG 00	149	STO IND 05
21	STO 02	2N+10	64	STO 08	107	NOP	150	X<>Y
22	STO 06		65	CF 01	108	RCL IND 06	151	RCL IND 06
23	FIX 0		66	XEQ 11	109	RCL IND 05	152	FS? 01
24	CF 29		67	ZENTER^	110	ZAVIEW	153	RCL 08
25	LBL 05		68	RCL 08	111	DSE 06	154	FS? 01
26	"IM^RE("	N	69	RCL 07	112	DSE 05	155	*
27	ARCL 03		70	P-R	113	GTO 10	156	+
28	"@)=?"		71	Z*	114	CF 21	157	FS? 00
29	PROMPT		72	ST- 03	115	SF 29	158	STO IND 06
30	STO IND 05		73	X<>Y	116	RTN	159	X<>Y
31	X<>Y		74	ST- 04	117	LBL 11	160	FS? 01
32	STO IND 06	N-1	75	ZRND	118	RCL 01	161	DSE 08
33	DSE 03		76	Z#0?	119	STO 05	162	LBL 02
34	X<>Y		77	GTO 01	120	RCL 02	163	DSE 06
35	DSE 06		78	FIX 0	121	STO 06	164	DSE 05
36	DSE 05		79	"FOUND ROOT#"	122	FC? 01	165	GTO 00
37	GTO 05		80	ARCL 00	123	GTO 13	166	END
38	RCL 03		81	AVIEW	124	E-3		
39	LBL 06		82	SF 00	125	ST+ 05		
40	"SOLVING..."		83	XEQ 11	126	LBL 13		
41	AVIEW		84	E	127	RCL IND 06		
42	SF 25		85	ST+ 05	128	RCL IND 05		
43	SF 99		86	ST+ 06	129	FC? 01		

Example.- Calculate the three roots of: $\mathbf{x^3 + x^2 + x + 1}$

ZPROOT -> "DEGREE=?"
3, R/S -> "IM^RE (3)=?"
0, ENTER^, 1, R/S -> "IM^RE (2)=?"
0, ENTER^, 1, R/S -> "IM^RE (1)=?"
0, ENTER^, 1, R/S -> "IM^RE (0)=?"
0, ENTER^, 1, R/S -> "SOLVING..."
-> "FOUND ROOT#3"
-> "SOLVING..."
-> "FOUND ROOT#2"
-> "SOLVING..."
-> "FOUND ROOT#1"
➔ -5,850E-14-j1 (that is, -i)
➔ 5,850E-14+j1 (that is, i)
➔ -1+j1,170E-13 (that is, -1)

Example:- Calculate the four roots of: $\mathbf{(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1}$

ZPROOT -> "DEGREE=?"
4, R/S -> "IM^RE (4)=?"
2, ENTER^, 1, R/S -> "IM^RE (3)=?"
2, CHS, ENTER^, 1, CHS, R/S -> "IM^RE (2)=?"
3, CHS, ENTER^, CHS, R/S -> "IM^RE (1)=?"
0, ENTER^, 1, R/S -> "IM^RE (0)=?"
0, ENTER^, 1, CHS, R/S -> "SOLVING..."
-> "FOUND ROOT#4"
-> "SOLVING..."
-> "FOUND ROOT#3"
-> "SOLVING..."
-> "FOUND ROOT#2"
-> "SOLVING..."
-> "FOUND ROOT#1"
➔ 1,698+J0,802 R/S
➔ -0,400-J0,859 R/S
➔ 0,358+J0,130 R/S
➔ -0,656-J0,073

The four solutions are:
z_1 = 1,698 + 0,802 j or: 1,878 <) 25,27
z_2 = -0,400 - 0,859 j or: 0,948 <)-114,976
z_3= 0,358 + 0,130 j or: 0,381 <) 9,941
z_4 = -0,,656 - 0,073 j or: 0,660 <)-173,676

Using V41's turbo mode (or another equivalent HP-41 emulator functionality) the execution time is largely reduced – to almost instantaneous response!

10.5 Solution to f(z)=0.

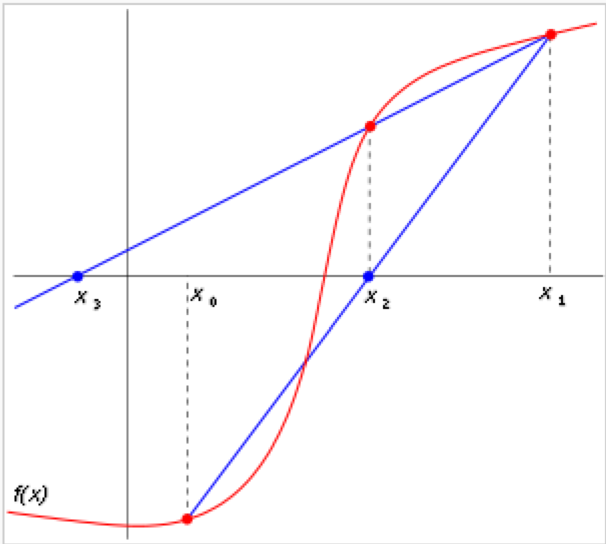
The final example uses the Secant Method to obtain roots of a complex equation, given two estimations of the solution. A general discussion on root-finding algorithms and is beyond the scope of this manual – this example is intended to show the capabilities of the 41Z module, in particular how programming with complex numbers becomes as simple as doing it for real numbers using the native function set.

See the following link for further reference on this subject (albeit just for real variable):
http://en.wikipedia.org/wiki/Secant_method

The secant method is defined by the recurrence relation:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

which will be calculated until there's no significant contribution to the new value – as determined by the function **Z=WR?**.



The first two iterations of the secant method. The red curve shows the function *f* and the blue lines are the secants.

Program listing:-

As it's the case with this type of programs, the accuracy of the solution depends of the display settings, and the convergence (i.e. likelihood to find a root) will depend on the initial estimations.

The program works with 8-digit precision, therefore will largely benefit from the turbo-mode settings on V41 to dramatically reduce the execution time.

1	LBL "ZSOLVE"	20	Z<>W	39	Z-
2	FS? 06	21	ZSTO (00)	40	Z*
3	GTO 06	22	XEQ IND 06	41	ZNEG
4	AON	23	ZSTO	42	ZRCL
5	"F. NAME=?"	24	2	43	1
6	PROMPT	25	LBL 01	44	Z+
7	AOFF	26	ZRCL	45	ZENTER^
8	ASTO 06	27	1	46	Z<>
9	PREC=?	28	XEQ IND 06	47	1
10	PROMPT	29	ZREPL	48	Z=WR?
11	FIX IND X	30	Z<>	49	GTO 02
12	"Z1=? (Y^X)"	31	2	50	GTO 01
13	PROMPT	32	Z-	51	LBL 02
14	ZENTER^	33	Z#0?	52	FC? 06
15	"Z2=? (Y^X)"	34	Z/	53	FIX 3
16	PROMPT	35	ZRCL	54	FC? 06
17	LBL 06	36	1	55	ZAVIEW
18	ZSTO	37	ZENTER^	56	END
19	1	38	Z<> (00)		

User flag 06 is for subroutine usage: when set, the data input will be skipped. In that case the relevant data is expected to be in the appropriate registers, as follows:

CR0 = Initial estimation z1,
Cr1 = initial estimation z2
R06 = Function's name,
FIX set manually to required precision.

Example.- Calculate one root of the equation: **SINH(z) + z^2 + pi = 0**

Which we easily program using 41Z functions as follows:
LBL "ZT", **ZSINH, LASTZ, Z^2, Z+,** PI, +, END.

Using the initial estimations as z0=0, and z1=1+i, we obtain:
Root = -0,27818986 + j 1,81288037

Example:- Calculate two roots of the equation: **e^(z) = z**
programmed as follows: LBL "ZE", **ZEXP**, **LASTZ**, **Z-**, END

using the estimations: {z0=-1-j & z1=1+j} - note that both roots are conjugated!

Root1 = 0,3181315 + j 1,3372357
Root2 = 0,3181315 - j 1,3372357

Example:- Calculate the roots of the polynomials from section 10.1 and 10.3:

P2 = (1+i)*z^2 + (-1-i)*z + (1-i) - re-written as: z[(-1-i)-z(1+i)} + (1-i)
P3 = z^3 + z^2 + z + 1 - re-written as: z[1+z(1+z)] +1
P4 = (1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z – 1
- re-written as: z{1+z[(3-3i)-z[(1+2i)-z(1+2i)]]}-1

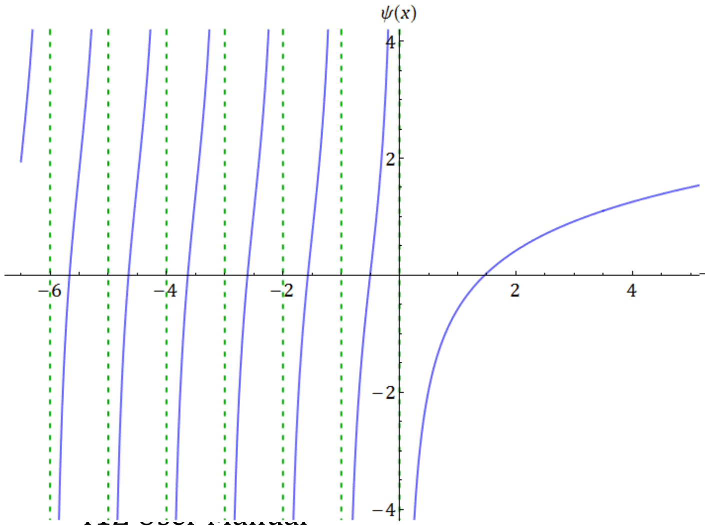
Use the following estimations for the P4 example:-

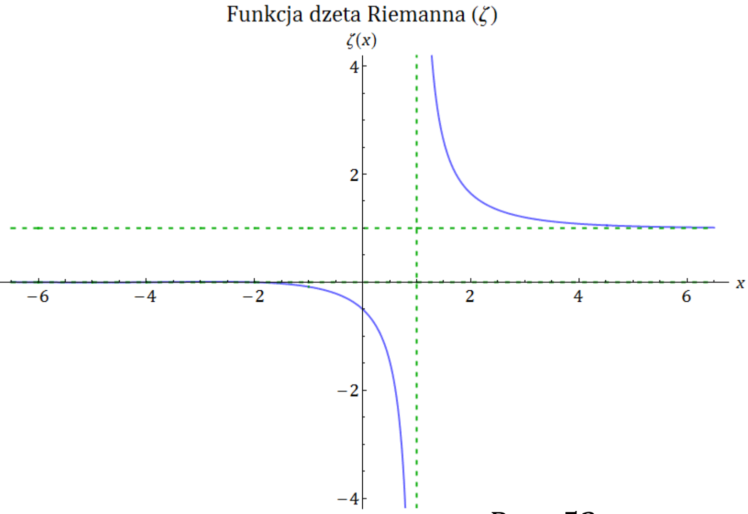
{z0=-1-j ; z1=1+j} for root #1, {z0=1+j ; z1=2+2j} for root #2,
{z0=-2j ; z1= 2j} for root #3, {z0= 4j ; z1= 5j} for root #4

And programmed as follows:

$(1+i)*z^2 + (-1-i)*z + (1-i) = 0$		$(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1$			
1	LBL "Z2"	1	LBL "Z4"	1	LBL "Z4"
2	ZREPL	2	ZREPL	2	ZREPL
3	1	3	2	3	4
4	ENTER^	4	ENTER^	4	Z^X
5	Z*	5	1	5	ZENTER^
6	ZENTER^	6	Z*	6	2
7	-1	7	LASTZ	7	ENTER^
8	ENTER^	8	Z-	8	1
9	Z+	9	Z*	9	Z*
10	Z*	10	ZENTER^	10	Z<>W
11	ZENTER^	11	-3	11	3
12	-1	12	ENTER^	12	Z^X
13	ENTER^	13	CHS	13	ZENTER^
14	CHS	14	Z+	14	-2
15	Z+	15	Z*	15	ENTER^
16	END	16	1	16	-1
		17	+	17	Z*
$z^3 + z^2 + z + 1$		18	Z*	18	Z+
1	LBL "Z3"	19	1	19	Z<>W
2	ZREPL	20	-	20	Z^2
3	1	21	END	21	ZENTER^
4	+			22	-3
5	Z*	*Note the usage of stack-lifting*		23	ENTER^
6	1	*functions to separate entries*		24	CHS
7	+	*(LASTZ and ZENTER^)*		25	Z*
8	Z*			26	Z+
9	1			27	Z+
10	+			28	1
11	END			29	-
				30	END

Lastly, a few other excellent programs written by Jean-Marc Baillard address the general solution to the equation f(z)=0. They don't use functions from the 41Z module, but are mentioned here for their obviously close related content. The programs can be found at the following link: http://www.hpmuseum.org/software/41/41cmpxf.htm





10.5.1. Application example;- Using ZSOLVE to calculate the Lambert W function.

In this example we see a few techniques applied together, combining the capabilities of the 41Z in a convenient way. The solution is a direct application of the definition, requiring very simple extra programming – albeit with the logical slow performance.

The Lambert W function is given by the following functional equation:

z = W(z) e^W(z)^, for every complex number *z*.

Which cannot be expressed in terms of elementary functions, but can be properly written with the following short program:

1	LBL "ZWL"
2	ZSTO
3	4
4	ZLN
5	ZENTER^
6	E
7	+
8	SF 06
9	"*WL"
10	ASTO 06
11	ZSOLVE
12	ZAVIEW
13	RTN
14	LBL "*W"
15	ZEXP
16	LASTZ
17	Z*
18	ZRCL
19	4
20	Z-
21	END

The complex value is expected to be in the **Z** complex stack level, and X,Y registers upon initialization. Set the FIX manually for the required precision.

Because **ZSOLVE** uses all the complex stack levels and registers 0 to 6, the argument is saved in the complex register 4 – corresponding to real registers 8 and 9, thus a SIZE 10 or higher is required (see register correspondence map below).

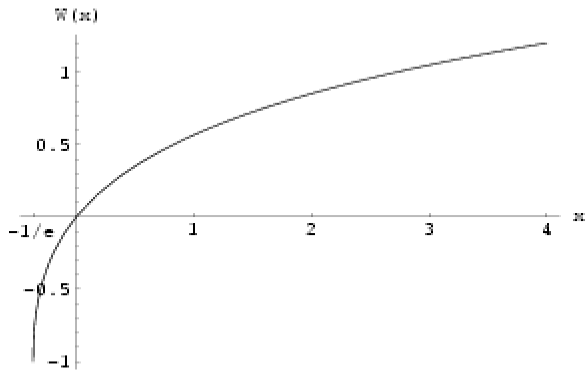
We solve for W(z)=z, using as the function initial estimations the logarithm of the same argument and the same point plus one, perhaps not a refined choice but sufficient to ensure convergence in the majority of cases. Some calculated values are:

$$W(0) = 0$$
$$W(1) = \Omega \approx 0.56714329\ldots$$
$$W(e) = 1$$
$$W(-1) \approx -0.31813 - 1.33723i$$

This example is not meant to compete with a dedicated program using an iterative algorithm, yet it showcases the versatility of the approach. The obvious speed shortcomings are diminished when ran on modern emulators like V41.

The Taylor series of W_0 around 0 is given by:

$$W_0(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} x^n$$



Another technique (somehow a brute-force approach) would employ this definition to calculate successive terms of the summation until their contribution to the sum is negligible. This method would only be applicable within the convergence region.

See the following links for further references on the Lambert W function:
http://en.wikipedia.org/wiki/Lambert_W_function
http://mathworld.wolfram.com/LambertW-Function.html

10.6 Bessel functions.

This last section represents an interesting "*tour de force*" within the 41Z module – taking the humble 41 system to the realm of true high-level math. Use it or leave it, it's all a matter of choice – but programming techniques and valid algorithms are always interesting, despite its obvious speed shortcomings.

Index	Function	Description	
1	**ZJBS**	Complex Bessel J function	First kind
2	**ZIBS**	Complex Bessel I function	First kind
3	**ZBS**	Subroutine for J and I	First & Second Kind
4	**ZKBS**	Complex Bessel K function	Second kind
5	**ZYBS**	Complex Bessel Y function	Second kind
6	**ZBS2**	Subroutine for K and Y	Second Kind
7	**EIZ/IZ**	Spherical Hankel first kind order zero	
8	**ZSHK1**	Spherical Hankel first kind	
7	**ZSHK2**	Spherical Hankel second kind	

See the paper "*Bessel functions on the 41 with the SandMath Module*" by the author, for an extensive description of the (real-number) Bessel Functions on the 41 system. In fact, following the "*do it as it's done with real numbers*" standard philosophy of the 41Z module, the complex versions of these programs are very similar to those real-number counterparts described in said paper.

The formulae used are as follows:

$$\mathbf{J(n,z) = \Sigma \{U_k \mid k=1,2,\dots\} * (z/2)^n / \Gamma(n+1)}$$
$$U(k) = -U(k-1) * (z/2)^2 / k(k+n)$$
$$U(0) = 1$$

$$\mathbf{Y_n(x) = [\, J_n(x) \cos(n\pi)) - J_{-n}(x)\,] / \sin(n\pi))}$$
$$\mathbf{K_n(x) = (\pi/2)\, [\, I_{-n}(x) - I_n(x)\,) / \sin(n\pi))\,]}$$
$$n \# \dots -3 ; -2 ; -1 ; 0 ; 1 ; 2 ; 3 ..$$

Like for the real case, there are two auxiliary functions, **ZBS** and **ZBS2**, to perform intermediate calculations used by the main programs: **ZJBS**, **ZIBS** (first kind), and **ZYBS**, **ZKBS** (second kind). Other auxiliary functions are:

- **GEUZ** – Euler's gamma constant – and
- **HARM**, to obtain the harmonic number of a given integer:
  H(n) = SUM [1/k] |k=1,2...n (*)

The expressions used to calculate the results are different for integer orders (remember the singularities of Gamma), requiring special branches of the main routines. For that reason two other functions have been added to the 41Z as follows:

- **ZINT?**, to determine integer condition, and
- **ZCHSX**, to simplify calculation of z*(-1)^k

Both the function order and the argument are complex numbers, which are expected to be on complex stack levels W (order) and Z (argument) prior to the execution of the function. The result is placed on the Z-level complex stack.

Below are the program listings for each particular case.-

a) *Bessel Functions of the first kind.*

1	LBL ZJBS		48	Z*	n
2	CF 00		49	ZRCL 00	n
3	GTO 00		50	RCL M	k
4	LBL ZIBS		51	+	n+k
5	SF 00		52	LASTX	k
8	LBL 00		53	ST* Z	k(n+k)
8	CF 01		54	*	
8	Z<>W		55	Z/	
9	ZINT?	is n integer?	56	ZSTO 02	U(k)
10	XEQ 05		57	ZRCL 03	SUM(k-1)
11	Z<>W		58	Z+	SUM(k)
12	ZHALF	z/2	59	ZENTER^	
13	XROM "ZBS"		60	Z<> 03	SUM(k-1)
14	FS? 01	n integer	61	Z=W?	
15	RCL 01		62	GTO 01	
16	FS? 01		63	E	
17	ZCHSX	J(-n, z) = (-1)^n J(n, z)	64	ST+ M	k=k+1
18	LBL 04		65	GTO 02	
19	ZAVIEW		66	LBL 01	
20	RTN		67	ZRCL 00	n
21	LBL 05		68	INCX	(n+1)
22	X<0?	n<0?	69	CF 02	
23	SF 01		70	X<0?	
24	ABS		71	SF 02	
25	RTN		72	X<0?	
26	LBL "ZBS"		73	ZNEG	-z
27	Z#0?		74	ZGAMMA	
28	GTO 00		75	FC? 02	
29	Z=W?		76	GTO 00	
30	E		77	LASTZ	-z
31	GTO 04		78	ZGNGZ	
32	LBL 00		79	Z<>W	
33	-ZSTACK	running…	80	Z/	
34	ZSTO 01	(z/2)	81	LBL 00	
35	Z<>W	n	82	Z/	
36	ZSTO 00	n	83	ZRCL 01	(z/2)
37	E	1	84	ZRCL 00	n
38	ZREAL	1+J0	85	W^Z	(z/2)^n
39	ZSTO 02	1+J0	86	Z*	
40	ZSTO 03	1+J0	87	END	
41	STO M	k=1			
42	LBL 02		CR00 - n		
43	ZRCL 01		CR01 - Z/2		
44	Z^2	(z/2)^2	CR02 - Uk		
45	ZRCL 02	Uk-1	CR03 - SUM		
46	FC? 00		CR04 - result		
47	ZNEG				

Examples:- Calculate JBS(1+i, -1-i) and IBS(-0.5+i; 1-0,5i)

1, ENTER^, **ZENTER^**, **ZNEG**, **ZJBS** --> -8,889 + j 2,295
1, ENTER^, 0,5, CHS, **ZENTER^**, ENTER^, 1, **ZIBS** --> 3,421 + j 1,178

b) <u>*Bessel functions of the second kind.*</u>

LBL "ZB1"	**SUM{f(n,x)}**
CLZ	
ZSTO 02	*Jn / In*
ZSTO 04	*SUM*
STO 01	*k=0*
LBL 02	
XEQ 10	*summing term*
Z=0?	*x=0?*
GTO 01	*ignore term*
ZRCL 04	*S(k-1)*
Z+	*S(k)*
ZENTER^	
Z<> 04	
Z=W?	*are they equal?*
RTN	*Final result(s)*
LBL 01	
E	*increase index*
ST+ 01	*k=k+1*
GTO 02	
LBL 10	***Function to Sum***
ZRCL 01	*x/2*
RCL 01	*k*
ST+ X	*2k*
RCL 00	*n*
+	*2k+n*
Z^X	*(x/2)^(2k+n)*
ZENTER^	
RCL 01	*k*
FACT	*k!*
LASTX	*k*
RCL 00	*n*
+	*k+n*
FACT	*(k+n)!*
*	*k! * (k+n)!*
ZREAL	
Z/	*k-th. Term*
FS? 00	***is it Kn?***
GTO 00	
RCL 01	*k*
ZCHSX	*[term] * (-1)^k*
LBL 00	
Z<> 02	***ZST+ 02***
ZRCL 02	
Z+	***f(k) + SUM(k-1)***
Z<> 02	Jn / In
ZENTER^	
RCL 01	*k*
HARMN	*H(k)*
LASTX	*k*
RCL 00	*n*
+	*k+n*
HARMN	*H(k+n)*
+	*H(k)+H(k+n)*
ZREAL	
Z*	
END	

LBL "ZB2"	**SUM{g(n,x)}**
CLZ	
ZSTO 03	*reset partial SUM*
RCL 00	*ABS(n)*
X=0?	*n=0?*
RTN	*skip it*
DECX	
E3	
/	*0,00(n-1)*
STO 08	
LBL 05	
ZRCL 01	*x/2*
RCL 08	*k,00(n-1)*
INT	
STO 01	*k*
ST+ X	*2k*
RCL 00	*n*
-	*2k-n*
Z^X	*(x/2)^(2k-n)*
RCL 00	*n*
RCL 01	*k*
-	*n-k*
DECX	*n-k-1*
FACT	*(n-k-1)!*
RCL 01	*k*
FACT	*k!*
/	*(n-k-1)! / K!*
ST* Z	
*	*[**]*
FC? 00	***is it Yn?***
GTO 00	
RCL 01	*k*
ZCHSX	*(-1)^k * term*
LBL 00	
ZRCL 03	
Z+	
ZSTO 03	
ISG 08	
GTO 05	*(k+1),00(n-1)*
ZRCL 03	
FC? 00	is it Yn?
RTN	
RCL 00	n
ZCHSX	*SUM*(-1)^n*
END	

Note: functions DECX and INCX can be replaced by standard FOCAL sequences:

DECX = 1, -
INCX = 1, +

1	LBL "ZYBS"	Integer Index	47	LBL 05	integer orders
2	CF 00		48	CF 01	
3	GTO 00		49	X<0?	negative
4	LBL "ZKBS"		50	SF 01	
5	SF 00		51	ABS	
6	LBL 00		52	STO 00	
7	ZHALF		53	XROM "ZB2"	
8	ZSTO 01	(z/2)	54	ZNEG	-[SUM*(-1)^n]
9	Z<>W		55	ZSTO 03	
10	ZINT?		56	XROM "ZB1"	to obtain both!
11	GTO 05		57	ZRCL 03	
12	Z<>W		58	Z<>W	
13	XROM "ZBS"		59	Z-	
14	FS? 00		60	ZRCL 01	x/2
15	GTO 00		61	ZLN	Ln(x/2)
16	ZRCL 00		62	GEU	g
17	PI		63	+	g+Ln(x/2)
18	ST* Z		64	ZRCL 02	J(n,x) or I(n,x)
19	*		65	Z*	[ ]*J/I(n,x)
20	ZCOS		66	ZDBL	
21	Z*		67	Z+	K(n,x)/Y(n,x)
22	LBL 00		68	FC? 00	is it Yn?
23	ZSTO 04		69	GTO 04	FINAL STEPS
24	ZRCL 00	n	70	RCL 00	n
25	ZNEG	-n	71	INCX	(n+1)
26	ZRCL 01	(z/2)	72	ZCHSX	K(n,x)* (-1)^(n+1)
27	XROM " ZBS"		73	ZHALF	
28	ZRCL 04		74	GTO 03	Exit
29	Z<>W		75	LBL 04	Yn
30	Z-		76	PI	
31	ZRCL 00	-n	77	ST/ Z	
32	ZNEG	n	78	/	
33	PI		79	FC? 01	negative index?
34	ST* Z		80	GTO 03	Exit
35	*		81	RCL 00	n
36	ZSIN		82	ZCHSX	
37	Z/		83	LBL 03	
38	FC? 00		84	ZSTO 03	
39	GTO 03	Exit	85	ZAVIEW	
40	PI		86	END	
41	2				
42	/				
43	CHS				
44	ST* Z				
45	*				
46	GTO 03	Exit			

The formulae used for integer orders are as follows:

$$\pi\, Y_n(x) = 2[\gamma + \mathrm{Ln}\, x/2]\, J_n(x) - \Sigma\, (-1)^k f_k(n,x) - \Sigma\, g_k(n,x)$$

$$(-1)^{n+1}\; 2\, K_n(x) = 2\,[\gamma + \mathrm{Ln}\, x/2]\, I_n(x) \;-\Sigma\, f_k(n,x) - \;(-1)^n\, \Sigma\, (-1)^k\, g_k(n,x)$$

$g_k(n,x) = (x/2)^{2k-n}\; [(n-k-1)! \,/\, k!]$; k=0,2,...(n-1)

$f_k(n,x) = (x/2)^{2k+n}\; [H(k) + H(n+k)] \,/\, [k!\,(n+k)!]$; k=0,1,2,.....

Example:- Calculate KBS(-0.5+i; 1-0,5i)
1, ENTER^, 0,5, CHS, **ZENTER^**,
ENTER^, 1, **ZKBS** → 0,348 + j 0,104

Example:- Calculate YBS(-1,-1)

0, ENTER^, 1, CHS, **ZENTER^**,
ZYBS → - 0,781 + j 0,880

This last example shows how even real arguments can yield complex results.

Example.- Calculate JBS and IBS for (1+2i, -1-3i)

2, ENTER^, 1, **ZENTER^**
3, CHS, ENTER^, 1, CHS, **ZIBS** → 35,813 - j 191,737

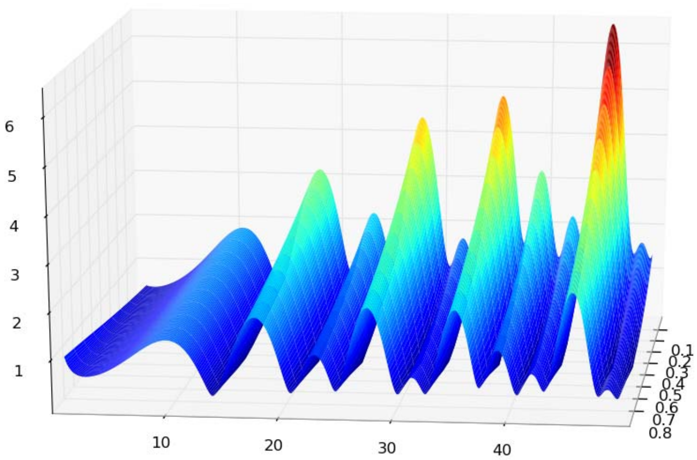
2, ENTER^, 1, **ZENTER^**
3, ENTER^, 1, **ZNEG, ZJBS** → - 257,355 - j 12,633

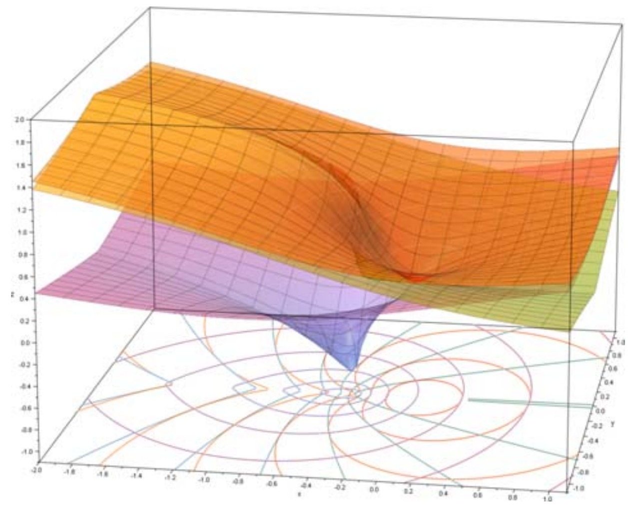
Note: *Using the Complex Keyboard shortcuts* the Bessel function group can be accessed pressing SHIFT when the NEXT indicator is shown, as per the following sequence:

Z, **Z**, SHIFT, SHIFT -> then I, J, for ZJBS and ZJBS or D, E for ZKBS and ZYBS.

The same group can be used to access **ZWL** (Complex Lambert) and **EIZ/IZ**, the Spherical Hankel function of first kind and order zero.

, then SHIFT:



Appendix 1.- Complex Buffer functions

This appendix lists the buffer handling functions included in the 41Z, and thus are not related to the Complex Number treatment per se. This set is only useful to diagnose problems or to bypass the normal execution of the module's "standard" functions, therefore its use is not recommended to the casual user (i.e. do it at your own risk!).

Function	Description	Input	Output
-HP 41Z	Initializes Z Buffer	None	Buffer created
CLZB	Clears Z buffer	None	Buffler cleared
L1=XY?	Is L1 equal to XY?	None	Y/N, skip if false
L1<>L _	**Swap L1 & Level**	**Level# as suffix**	levels exchanged
L1<>LX	Swap L1 & Level	level in X	levels exchanged
L2=ZT?	Is L2 equal to ZT?	None	Y/N, skip if false
L2>ZT	Copies L2 into ZT	None	L2 copied to ZT
LVIEW _	**View Level**	**Level# as suffix**	*Transposed value!*
LVIEWX	View level by X	level in X	*Transposed value!*
PREMON	Copies XY into L0 and finds Zbuffer	Re(z) in X; Im(z) in Y	none
PSTMON	Copies XY into L1 and synch's up	Complex stack Z	Re(z) in X; Im(z) in Y
RG>ZB _ _	**Copies registers to Z buffer**	**Reg# as suffix**	data copied from registers
ST>ZB	Copies real stack to L1 & L2	None	stack copied to buffer
XY>L _	**Copies XY into Level**	**Level# as suffix**	XY copied to LEVEL
XY>L0	Copies XY into L0	Re(z) in X; Im(z) in Y	XY copied to L0
XY>L1	Copies XY into L1	Re(z) in X; Im(z) in Y	XY copied to L1
ZB>RG _ _	**copies buffer to registers**	**Reg# as suffix**	data copied to registers
ZB>ST	Copies L1 & L2 into real stack	None	buffer copied to Stack
ZBDROP	Drops Z buffer one level	None	levels dropped
ZBHEAD	Z buffer Header info	None	header register in ALPHA
ZBLIFT	Lifts Z buffer one level	None	buffer lifted
ZBSHOW	Shows Z Buffer	None	shows header & all levels

(*) Items highlighted in yellow indicate prompting functions.

Buffer layout. The complex buffer has 5 levels, labelled L0 to L4; that's 10 memory registers plus the header and footer registers – for a total of 12 registers. The function names in this group use the Level number (L0 to L4) to identify each level, and not the **U, V, W**, and **Z** notation employed in previous sections of the manual.

	Buffer Layout		Buffer Details
b11	*non-zero*	-	*The buffer has 12 memory registers*
b10	L4 (U)	-	*Buffer registers are labeled b0 to b11*
b9		-	*Header is located at the bottom*
b8	L3 (V)	-	*A non-zero register is at the top*
b7		-	*Each Level uses two buffer registers*
b6	L2 (W)	T	*Levels are labeled L0 to L4*
b5		Z	
b4	L1 (Z)	Y	
b3		X	
b2	L0 (S)	L	
b1		-	
b0	Header	-	

The buffer header (b0 register) is placed at the lowest memory address. It contains the buffer id#, its size, and its initial address (when it was first created – no updates if it's re-allocated later on).

Buffer creation is done automatically by the 41Z module upon power on (when the 41 awakes from deep sleep), using the corresponding poll point in the module. The contents of the real stack registers XYZT is copied into the buffer levels L1 & L2 upon initialization.

The buffer is maintained by the 41 OS, which handles it when modifying the layout of main memory – either changing the SIZE settings, or modifying the user key assignments. The buffer id# is 8, and thus should be compatible with any other memory buffer that uses a different id# (an example of which are the TIMER alarms).

Should for any reason the buffer is damaged or erased (like when using the function **CLZB**), the message "NO Z-BUFFER" would appear when trying to execute any of the 41Z module functions. *To manually re-create the complex buffer* simply execute the first function in the module, "**–HP 41Z**" - either by using XEQ or the Complex Keyboard sequence "**Z**, SHIFT, **Z**". This requires at least 12 memory registers to be available or the error message "NO ROOM" will be shown.

Because the buffer can be dynamically re-allocated by the 41 OS upon certain circumstances, it's not possible to store its address to be reused by the functions. *Every 41Z function would first seek out the buffer address prior to proceeding with its calculation*. Fortunately this takes very little overhead time.

Buffer synchronization with the appropriate real-stack levels is also performed automatically by the 41Z functions, as follows:

- In the input phase (pre-execution), monadic functions will copy the XY contents into level L1 prior to executing their code. Dual functions will do the same for the second argument **Z**, and will use the current contents of the L2 level as first argument **W**.

- In the output phase (post-execution) the results will be placed in the complex buffer levels and then copied to the real stack registers as appropriate: XY for monadic functions, and XZYT for dual functions.

That's the reason why the real stack should just be considered as a *scratch pad* to prepare the data (like doing math on the real values), as only levels X,Y will be used. You must use **ZENTER^** to push the **W** argument into the complex level L2. In other words: real stack registers T,Z will be ignored!

The same consideration applies when performing chain calculations: because there's no automated complex stack lift, *the result of a monadic function would be overwritten by the subsequent input unless it is first pushed into the complex stack*, using **ZENTER^** *or another 41Z function that does stack lift.*

Example: Calculate Ln(1+i) + (2-i)

The following sequence use the direct data entry, entering Im(z) first.
1, ENTER^, **ZLN**, **ZENTER^**, 1, CHS, ENTER^, 2, **Z+** -> 2,347-j0,215

*Some functions perform stack lift by default, and thus **ZENTER^** is not required before them*.

They are as follows:

- **LASTZ**
- **ZRCL _ _**
- **ZREAL^** (also when using the complex real keypad, Z plus digit key)

- **ZIMAG^** (also when using the complex imaginary keypad, Z, radix, plus digit key)
- **^IM/AG _**

The following sequence uses natural data entry - entering Re(z) first - as an alternative method for the previous example. Note that because **^IMG** does stack lift, it's not necessary to use **ZENTER^**

1, **^IMG,** 1, R/S**, ZLN,** 2, **^IMG,** 1, CHS, R/S**, Z+** -> 2,347-j0,215

Buffer synchronization with the real stack registers can be tested and forced using the following functions in this group:

L1=XY? - Tests for the first buffer level and XY registers	
XY>L1 - Copies X,Y into level L1	
L2=ZT? - Tests for second buffer level and Z,T registers	
L2>ZT - Copies L2 into registers Z,T	
ST>ZB - Copies real stack XYZT to buffer levels L1 & L2	
ZB>ST - Copies L1 & L2 to the real stack XYZT	

To dump the complete contents of the complex buffer into memory registers and back you can use these two complementary functions:

ZB>RG _ _ - Copies complex buffer to memory registers
RG>ZB _ _ - Copies memory registers to complex buffer

Note that **RG>ZB** won't check for valid header data, thus it expects the contents to be correct – like with a previously execution of **ZB>RG**. Remember that the header register is a non-normalized number (NNN), thus do not recall it using RCL.

Other functions to manipulate the contents of the buffer levels are:

L1<>L _ - swaps buffer level L1 and level given by prompt
L1<>LX - swaps buffer level L1 and level input in X
XY>L0 - copies registers X,Y into buffer level L0 (used to save arguments into LastZ)
XY>L _ - copies registers X,Y into buffer level given by prompt
ZBDROP - drops contents of complex buffer one level (used during **ZRDN**)
ZBLIFT - lifts contents of complex buffer one level (used by **ZRUP**, **ZENTER^** and others)

All these functions act on the complex buffer, but will not display the "resulting" complex number (i.e. will not trigger **ZAVIEW** upon completion). To see (view) the contents of the buffer levels without altering their position you can use the following functions:

LVIEW _ - prompts for level number (0 – 4)
LVIEWX - expects level number in X
ZBSHOW - lists the contents of all buffer levels
ZBHEAD - shows in Alpha the decoded buffer header

Note that all complex level contents will be shown transposed, that is: Im(z) + j Re(z).

Finally, the other two functions are auxiliary mainly used to perform action between the two lower and upper 4k-pages within the 41Z module: (*)

PREMON	- Finds Z Buffer address, Copies XY into L0 and checks X,Y for ALPHA DATA
PSTMON	- Copies the Z complex level into X.Y

(*) *Note: FAT entries for these two functions were removed in newer versions of the module.*

Because of its relevance and importance within the 41Z module, the following section lists the buffer creation and interrogation routines – pretty much the heart of the implementation. Consider that they are called at least twice every time a function is executed and you'll appreciate their crucial role in the whole scheme!

	CHKBUF	CHKBUF	A998	04E	C=0 ALL	
	CHKBUF	CHKBUF	A999	130	LDI S&X	
	CHKBUF	CHKBUF	A99A	008	CON: 8	*Buffer id# in C(14)*
	CHKBUF	CHKBUF	A99B	23C	RCR 2	*ID# to C(12)*
	CHKBUF	CHKBUF	A99C	35C	PT= 12	
	CHKBUF	CHKBUF	A99D	130	LDI S&X	
	CHKBUF	CHKBUF	A99E	0BF	CON: 191	*Fisrt possible reg -1*
	CHKBUF	CHKBUF	A99F	10E	A=C ALL	
	CHKBUF	**CB10**	A9A0	166	A=A+1 S&X	*Increase address*
	CHKBUF	**CB20**	A9A1	046	C=0 S&X	*Select Chip 0*
	CHKBUF	CHKBUF	A9A2	270	RAM SLCT	
	CHKBUF	CHKBUF	A9A3	378	READ 13(c)	*.END.*
	CHKBUF	CHKBUF	A9A4	306	?A<C S&X	*did we reach the .END. Chainhe*
	CHKBUF	CHKBUF	A9A5	**3A0**	**?NC RTN**	*yes -> Not Found*
	CHKBUF	CHKBUF	A9A6	0A6	A<>C S&X	
	CHKBUF	CHKBUF	A9A7	270	RAM SLCT	*Candidate address*
	CHKBUF	CHKBUF	A9A8	0A6	A<>C S&X	
	CHKBUF	CHKBUF	A9A9	038	READ DATA	*Candidate Value*
	CHKBUF	CHKBUF	A9AA	2EE	?C#0 ALL	Carry if not empty register
	CHKBUF	CHKBUF	A9AB	**3A0**	**?NC RTN**	*Zero reg ->Not Found*
	CHKBUF	CHKBUF	A9AC	23E	C=C+1 MS	not zero, keep searching
	CHKBUF	CHKBUF	A9AD	***39F***	***JC -13***	*KAR*
	CHKBUF	CHKBUF	A9AE	362	?A#C @PT	*IS this IO Buffer?*
	CHKBUF	CHKBUF	A9AF	***037***	***JC +06***	*NO*
	CHKBUF	CHKBUF	A9B0	1B0	POP ADR	*YES*
	CHKBUF	CHKBUF	A9B1	23A	C=C+1 M	*Return to (P+2)*
	CHKBUF	CHKBUF	A9B2	170	PUSH ADR	
	CHKBUF	CHKBUF	A9B3	038	READ DATA	*Return with Header in C*
	CHKBUF	CHKBUF	A9B4	**3E0**	**RTN**	***and BuffAdr in A***
	CHKBUF	**CB30**	A9B5	0FC	RCR 10	*Skip Buffer*
	CHKBUF	CHKBUF	A9B6	056	C=0 XS	
	CHKBUF	CHKBUF	A9B7	146	A=A+C S&X	
	CHKBUF	CHKBUF	A9B8	***34B***	***JNC -23d***	*[CB20]*
	INITIALIZE	*Header*	A9B9	*09A*	***"Z"***	
	INITIALIZE	*Header*	A9BA	*031*	***"1"***	
	INITIALIZE	*Header*	A9BB	*034*	***"4"***	
	INITIALIZE	*Header*	A9BC	*020*	***" "***	
	INITIALIZE	*Header*	A9BD	*010*	***"P"***	***Programmable!***
	INITIALIZE	*Header*	A9BE	*008*	***"H"***	
	INITIALIZE	*Header*	A9BF	*02D*	***"-"***	
	INITIALIZE	**INITIALIZE**	**A9C0**	*379*	*PORT DEP:*	*Check for buffer*
	INITIALIZE	INITIALIZE	A9C1	*03C*	*XQ*	*Get its address if exists*
			A9C2	*198*	*->A998*	*[CHKBUF]*
b11	***non-zero***		A9C3	***073***	***JNC +14d***	*Not Found - Create it !!*
b10	**L4**	-	A9C4	*379*	*PORT DEP:*	***0.- write XYZT into [b3-b6]***
b9		-	A9C5	*03C*	*XQ*	*to initialize L1 & L2*
b8	**L3**	-	A9C6	*186*	*->A986*	*[SYNCH2]*
b7		-	**A9C7**	**130**	**LDI S&X**	*A holds b7 address*
b6	**L2**	***T***	A9C8	**004**	**CON: 4**	*adds 4 to it*
b5		***Z***	A9C9	**206**	**C=C+A S&X**	***b11 addr***
b4	**L1**	***Y***	A9CA	270	RAM SELECT	*non-zero the last buffer reg*
b3		***X***	A9CB	2F0	WRIT DATA	*this should do it*
b2	**L0**	-	A9CC	2CC	?FSET 13	*Exit if PRG Running*
b1		***L***	A9CD	360	?C RTN	
b0	**Header**		A9CE	*3AD*	*PORT DEP:*	*Show X,Y*
			A9CF	*08C*	**GO**	
	INITIALIZE	INITIALIZE	A9D0	*000*	*->AC00*	*[ZAVIEW]*

Notice how we finish with ZAVIEW to show the current complex number in the stack upon buffer creation. [CHKBUF] does not create the buffer, but reads its address into register A and the content of the header into register C. The following section shows the actual buffer creation snippets.

	BUFFER	CREATE	A9D1	0A6	A<>C S&X	*Create IO buffer 880C000…*
	BUFFER	CREATE	A9D2	158	M=C ALL	*First free reg. address (from .EN*
	BUFFER	CREATE	A9D3	04E	C=0 ALL	
	BUFFER	CREATE	A9D4	270	RAM SLCT	*Select Chip0*
	BUFFER	CREATE	A9D5	*285*	*?NC XQ*	
	BUFFER	CREATE	A9D6	*014*	*->05A1*	*[MEMLFT]*
	BUFFER	CREATE	A9D7	106	A=C S&X	*number of "free regs"*
	BUFFER	CREATE	A9D8	130	LDI S&X	*Must be at least 12 free regs.*
	BUFFER	CREATE	A9D9	***00C***	***CON: 12***	*(header + 5 complex stack leve*
	BUFFER	CREATE	A9DA	306	?A<C S&X	*Enough Memory?*
	BUFFER	CREATE	A9DB	***05F***	***JC +11d***	*[NORMER]*
	BUFFER	CREATE	A9DC	198	C=M ALL	First free reg. address (from .EN
	BUFFER	CREATE	A9DD	270	RAM SLCT	select buffer header
	BUFFER	CREATE	A9DE	106	A=C S&X	***buffer address in A S&X***
	BUFFER	CREATE	A9DF	2DC	PT= 13	
	BUFFER	CREATE	A9E0	210	LD@PT- 8	***Buffer id#***
	BUFFER	CREATE	A9E1	210	LD@PT- 8	***Buffer id#***
	BUFFER	CREATE	A9E2	010	LD@PT- 0	*Buffer size*
	BUFFER	CREATE	A9E3	***310***	***LD@PT- 12***	*Buffer size*
	BUFFER	CREATE	A9E4	2F0	WRIT DATA	***Store Header Reg***
	BUFFER	CREATE	A9E5	***2FB***	***JNC -33d***	*A9C4*
	BUFFER	**NORMER**	A9E6	*3C1*	*?NC XQ*	*Enable & Clear Disp*
	BUFFER	NORMER	A9E7	*0B0*	*->2CF0*	*[CLLCDE]*
	BUFFER	NORMER	A9E8	*3BD*	*?NC XQ*	*Message Line*
	BUFFER	NORMER	A9E9	*01C*	*->07EF*	*[MESSL]*
	BUFFER	NORMER	A9EA	00E	**"N"**	
	BUFFER	NORMER	A9EB	00F	**"O"**	
	BUFFER	NORMER	A9EC	020	**" "**	
	BUFFER	NORMER	A9ED	012	**"R"**	
	BUFFER	NORMER	A9EE	00F	**"O"**	
	BUFFER	NORMER	A9EF	00F	**"O"**	
	BUFFER	NORMER	A9F0	20D	**"M"**	
	BUFFER	NORMER	A9F1	***083***	***JNC +16d***	
	BUFFER	NOBUFER	A9F2	*3C1*	*?NC XQ*	*Enable & Clear Disp*
	BUFFER	NOBUFER	A9F3	*0B0*	*->2CF0*	*[CLLCDE]*
	BUFFER	NOBUFER	A9F4	*3BD*	*?NC XQ*	*Message Line*
	BUFFER	NOBUFER	A9F5	*01C*	*->07EF*	*[MESSL]*
	BUFFER	NOBUFER	A9F6	00E	**"N"**	
	BUFFER	NOBUFER	A9F7	00F	**"O"**	
	BUFFER	NOBUFER	A9F8	020	**" "**	
	BUFFER	NOBUFER	A9F9	01A	**"Z"**	
	BUFFER	NOBUFER	A9FA	02D	**"-"**	
	BUFFER	NOBUFER	A9FB	002	**"B"**	
	BUFFER	NOBUFER	A9FC	015	**"U"**	
	BUFFER	NOBUFER	A9FD	006	**"F"**	
	BUFFER	NOBUFER	A9FE	006	**"F"**	
	BUFFER	NOBUFER	A9FF	005	**"E"**	
	BUFFER	NOBUFER	AA00	212	**"R"**	
	BUFFER	NOBUFER	AA01	*3DD*	*?NC XQ*	
	BUFFER	NOBUFER	AA02	*0AC*	*->2BF7*	*[LEFTJ]*
	BUFFER	NOBUFER	AA03	108	SETF 8	
	BUFFER	NOBUFER	AA04	*201*	*?NC XQ*	*encp00, tell ptr, blink and set m*
	BUFFER	NOBUFER	AA05	*070*	*->1C80*	*[MSG105]*
	BUFFER	NOBUFER	AA06	*3ED*	*?NC GO*	*HALT execution*
	BUFFER	NOBUFER	AA07	*08A*	*-> 22FB*	*[ERR110]*

Remember that the buffer is created each time the calculator is turned on, and that it gets reallocated when key assignments or other buffers (like timer alarms) are made – yet it's possible that it gets "unsynchronized" or even lost altogether, and therefore the assignment to the **–HP 41Z** function as well.

Appendix 2. Complex Keyboard keymaps.

The following table shows the detailed key map supported by the ZKBRD complex keyboard function launcher.

Level					Function
I	II	III	IV	V	Name
Z	1/X				ZINV
Z	SQRT				ZSQRT
Z	LOG				ZLOG
Z	LN				ZLN
Z	X<>Y				Z<>W
Z	RDN				ZRDN
Z	SIN				ZSIN
Z	COS				ZCOS
Z	TAN				ZTAN
Z	XEQ				^IMG _
Z	STO				ZSTO _ _
Z	RCL				ZRCL _ _
Z	SST				Z<> _ _
Z	ENT^				ZENTER^
Z	CHS				ZNEG
Z	EEX				Z^X
Z	-				Z-
Z	+				Z+
Z	*				Z*
Z	/				Z/
Z	0-9				Z0-Z9
Z	R/S				ZAVIEW
Z	,	0-9			ZJ0-ZJ9
Z	Z	1/X			W^1/Z
Z	Z	SQRT			ZPSI
Z	Z	LOG			ZLNG
Z	Z	LN			e^Z
Z	Z	X<>Y			Z<>V
Z	Z	RDN			ZQRT
Z	Z	XEQ			ZIMAG^
Z	Z	STO			ZREAL^
Z	Z	RCL			Z/I
Z	Z	SST			CLSTZ
Z	Z	ENT^			ZRPL
Z	Z	EEX			Z^1/X
Z	Z	-			Z#W?
Z	Z	7			ZWDET
Z	Z	8			ZWDIST
Z	Z	9			ZWANG
Z	Z	+			ZREAL?
Z	Z	4			ZIN?
Z	Z	5			ZWCROSS
Z	Z	*			ZIMAG?
Z	Z	1			ZUNIT?
Z	Z	2			ZWLINE
Z	Z	/			Z#0?
Z	Z	0			ZOUT?
Z	Z	,			ZWDOT
Z	Z	Z			Z<>U

Level					Function
I	II	III	IV	V	Name
Z		ノ-			-HP 41Z
Z		Y^X			W^Z
Z		X^2			Z^2
Z		10^X			ZALOG
Z		e^X			ZEXP
Z		X<>Y			ZTRP
Z		RDN			ZRUP
Z		ASIN			ZASIN
Z		ACOS			ZACOS
Z		ATAN			ZATAN
Z		ASN			ZK?YN
Z		LBL			ZSIGN
Z		GTO			Z*I
Z		CAT			^IMG _
Z		ISG			ZCONJ
Z		RTN			X^Z
Z		CLX			CLZ
Z		X=Y?			Z=W?
Z		SF			ZNORM
Z		CF			ZMOD
Z		FS?			ZARG
Z		X<=Y?			Z=WR?
Z		BEEP			ZTONE
Z		P-R			ZREC
Z		R-P			ZPOL
Z		X>Y?			Z=I?
Z		FIX			ZRND
Z		SCI			ZINT
Z		ENG			ZFRC
Z		X=0?			Z=0?
Z		PI			ZGAMMA
Z		LASTX			LASTZ
Z		VIEW			ZVIEW _ _
Z			SIN		ZSINH
Z			COS		ZCOSH
Z			TAN		ZTANH
Z				SIN	ZASINH
Z				COS	ZACOSH
Z				TAN	ZATANH
Z	Z		SQRT		ZNXTNRT _
Z	Z		LN		ZNXTLN
Z	Z		SIN		ZNXTASN
Z	Z		COS		ZNXTACS
Z	Z		TAN		ZNXTATN
Z	Z			LOG	ZKBS
Z	Z			LN	ZYBS
Z	Z			COS	ZIBS
Z	Z			TAN	ZJBS
Z	Z			SIN	ZWL
Z	Z			SQRT	EIZ/IZ

Appendix 3.- Formula Compendium.

Elementary complex numbers and functions – By W. Doug Wilder.

$j=\sqrt{-1}=e^{j\frac{\pi}{2}}=1\angle 90^o$ $\quad j^2=e^{j\pi}=1\angle 180^o=-1$ $\quad -j=j^{-1}$

$Z=Re(Z)+jIm(Z)=x+jy=re^{j\theta}=r\angle\theta=r\cos\theta+jr\sin\theta$ $\quad r=|Z|=\sqrt{x^2+y^2}$ $\quad \theta=tan_4^{-1}(y/x)$

$\overline{Z}=Z^*=x-jy=re^{-j\theta}=r\angle-\theta$ $\quad Z+Z^*=2Re(Z)$ $\quad Z-Z^*=j2Im(Z)$

$(Z_1Z_2)^*=Z_1^*Z_2^*$ $\quad (Z_1Z_2^*)^*=Z_2Z_1^*$ $\quad (Z_1/Z_2)^*=Z_1^*/Z_2^*$ $\quad (Z_1+Z_2)^*=Z_1^*+Z_2^*$ $\quad (Z_1-Z_2)^*=Z_1^*-Z_2^*$

$|Z_1Z_2|=|Z_1||Z_2|$ $\quad |Z_1/Z_2|=|Z_1|/|Z_2|$ $\quad |Z_1Z_2^*|=|Z_1Z_2|$ $\quad r^2=|Z|^2=ZZ^*=x^2+y^2$

$|Z_1+Z_2|^2=(Z_1+Z_2)(Z_1+Z_2)^*=Z_1Z_1^*+Z_1Z_2^*+Z_2Z_1^*+Z_2Z_2^*=|Z_1|^2+2Re(Z_1Z_2^*)+|Z_2|^2$

$Z_1+Z_2=(x_1+x_2)+j(y_1+y_2)$ $\quad Z_1-Z_2=(x_1-x_2)+j(y_1-y_2)$ $\quad |Z_1+Z_2|\le|Z_1|+|Z_2|$

$Z_1Z_2=(x_1x_2-y_1y_2)+j(x_1y_2+y_1x_2)=r_1r_2\angle(\theta_1+\theta_2)=r_1r_2e^{j(\theta_1+\theta_2)}$ $\quad Re(1/Z^*)=Re(1/Z)$

$Z_1/Z_2=(x_1x_2+y_1y_2+j(y_1x_2-x_1y_2))/(x_2^2+y_2^2)=r_1/r_2\angle(\theta_1-\theta_2)$ $\quad Z^{-1}=(x-jy)/(x^2+y^2)=e^{-j\theta}/r$

$\overline{Z}_1Z_2=(x_1x_2+y_1y_2)+j(x_1y_2-y_1x_2)=r_1r_2\angle(\theta_2\text{-}\theta_1)=Z_1\bullet Z_2+jZ_1\times Z_2$ $\quad (Z_1,Z_2=2D\ vectors)$

$Z^2=x^2-y^2+j2xy=r^2e^{j2\theta}$ $\quad Z^{1/2}=r^{1/2}e^{j\theta/2}\ (principal)$

$\pi=3.14159\,26535\,89793\,23846\,264\ldots$ $\quad e^{Z+j2\pi n}=e^Z$ $\quad e=2.71828\,18284\,59045\,23536\,028\ldots$

$e^Z=e^xe^{jy}=e^x\angle y=e^x\cos y+je^x\sin y$ $\quad e^{Z_1}e^{Z_2}=e^{Z_1+Z_2}$ $\quad (e^{Z_1})^{Z_2}=e^{Z_1Z_2}\ (-\pi<\theta_1\le\pi)$

$e^{-Z}=1/e^Z$ $\quad e^{\ln Z}=Z$ $\quad e^{jZ}=cos(Z)+j\,sin(Z)$ $\quad e^Z=cosh(Z)+sinh(Z)=cos(jZ)-j\,sin(jZ)$

$\ln Z=\ln r+j\theta=\ln\sqrt{x^2+y^2}+j\,tan_4^{-1}(y/x)+j2\pi n$ $\quad e^{-jZ}/(-jZ)=(sinZ+j\,cosZ)/Z=h_0^{(2)}(Z)$

$Z_1\ln Z_2=\ln Z_2^{Z_1}$ $\quad \ln Z_1+\ln Z_2=\ln(Z_1Z_2)$ $\quad \ln 0=\infty$ $\quad \ln e^Z=Z$

$Z_2^{Z_1}=e^{Z_1\ln Z_2}$ $\quad \mathrm{Log}_aZ=\ln Z/\ln a$ $\quad \ln j=0+j\pi/2$ $\quad \ln 1=0$ $\quad \ln(-1)=0+j\pi$

$\frac{\partial}{\partial Z}Z^a=aZ^{a-1}$ $\quad \frac{\partial}{\partial Z}e^{aZ}=ae^{aZ}$ $\quad \frac{\partial}{\partial Z}a^Z=a^Z\ln a$ $\quad \frac{\partial}{\partial Z}\ln Z=\frac{1}{Z}$ $\quad \int e^{aZ}dZ=\frac{e^{aZ}}{a}$ $\quad \int\frac{dZ}{Z}=\ln Z$

$e^Z=1+\frac{Z}{1!}+\frac{Z^2}{2!}+\frac{Z^3}{3!}+\ldots$ $\quad \ln Z=\frac{2}{1}\left(\frac{Z-1}{Z+1}\right)+\frac{2}{3}\left(\frac{Z-1}{Z+1}\right)^3+\frac{2}{5}\left(\frac{Z-1}{Z+1}\right)^5+\ldots\ (Re(Z)\ge 0)$

$sin\,Z=(-j/2)(e^{jZ}-e^{-jZ})=(-j/2)(e^{jZ}-1/e^{jZ})$ $\quad sin^{-1}Z=-j\ln\left(jZ+\sqrt{1-Z^2}\right)$

$cos\,Z=(1/2)(e^{jZ}+e^{-jZ})=(1/2)(e^{jZ}+1/e^{jZ})$ $\quad cos^{-1}Z=-j\ln\left(Z+\sqrt{Z^2-1}\right)$

$tan\,Z=-j\frac{e^{jZ}-e^{-jZ}}{e^{jZ}+e^{-jZ}}=-j\frac{e^{j2Z}-1}{e^{j2Z}+1}$ $\quad tan^{-1}Z=-\frac{j}{2}\ln\left(\frac{1+jZ}{1-jZ}\right)$ $\quad tan_4^{-1}(Z_2/Z_1)=-j\ln\left(\frac{Z_1+jZ_2}{\sqrt{Z_1^2+Z_2^2}}\right)$

$\frac{\partial}{\partial Z}cos\,Z=-sin\,Z$ $\quad \frac{\partial}{\partial Z}sin\,Z=cos\,Z$ $\quad \int cos\,Z\,dZ=sin\,Z$ $\quad \int sin\,Z\,dZ=-cos\,Z$

$sin\,Z=Z-\frac{Z^3}{3!}+\frac{Z^5}{5!}-\frac{Z^7}{7!}+\ldots$ $\quad cos\,Z=1-\frac{Z^2}{2!}+\frac{Z^4}{4!}-\frac{Z^6}{6!}+\ldots.$

$sinh\,Z=(1/2)(e^Z-e^{-Z})=-j\,sin(jZ)$ $\quad sinh^{-1}Z=\ln\left(Z+\sqrt{Z^2+1}\right)$

$cosh\,Z=(1/2)(e^Z+e^{-Z})=cos(jZ)$ $\quad cosh^{-1}Z=\ln\left(Z+\sqrt{Z^2-1}\right)$

$tanh\,Z=\frac{e^Z-e^{-Z}}{e^Z+e^{-Z}}=\frac{e^{2Z}-1}{e^{2Z}+1}=-j\,tan(jZ)$ $\quad tanh^{-1}Z=\frac{1}{2}\ln\left(\frac{1+Z}{1-Z}\right)$

$cos\,Z=cos\,x\,cosh\,y-j\,sin\,x\,sinh\,y$ $\quad sin\,Z=sin\,x\,cosh\,y+j\,cos\,x\,sinh\,y$

Appendix 4.- Quick Reference Guide. The tables in the following four pages list all 41Z functions in alphabetical order.

#	Function	Description	Formula	Input	Output	Comments
1	**-HP 41Z**	***Initializes Complex Stack***	***Z=XY; W=ZT***	***none***	*Initializes Z buffer & ZAVIEW*	*runs on CALC ON*
2	**W^1/Z**	Complex Y^1/X	w^1/z = exp(Ln w / Z)	w in ***W***; z in ***Z*** (XY)	w^1/z in ***Z*** (XY)	*Drops Buffer*
3	**W^Z**	Complex Y^X	w^z = exp(z*Ln w)	w in ***W***; z in ***Z*** (XY)	w^z in ***Z*** (XY)	*Drops Buffer*
4	**X^Z**	Hybrid Y^X	a^Z = exp(z*Ln a)	x in X reg; z in Y,Z regs	x^z in ***Z*** (XY)	*does LastZ*
5	**Z+**	Complex addition	(x1+x2) + i (y1+y2)	w in ***W***; z in ***Z*** (XY)	w+z in ***Z*** (XY)	*Drops Buffer, LastZ*
6	**Z-**	Complex substraction	w-z = w + (-z)	w in ***W***; z in ***Z*** (XY)	w-z in ***Z*** (XY)	*Drops Buffer, LastZ*
7	**Z***	Complex multiplication	(x1*x2 - y1*y2) + i (x1*y2 + y1*x2)	w in ***W***; z in ***Z*** (XY)	w*z in ***Z*** (XY)	*Drops Buffer, LastZ*
8	**Z/**	Complex division	w/z = w * (1/z)	w in ***W***; z in ***Z*** (XY)	w/z in ***Z*** (XY)	*Drops Buffer, LastZ*
9	**Z^1/X**	Hybrid Y^X	z^1/n = r^1/n * exp(i*Arg/n)	x in X reg; z in Y,Z regs	z^1/x in ***Z*** (XY)	*does LastZ*
10	**Z^2**	Complex X^2	z^2 = r^2 * exp(2i*Arg)	z in ***Z*** (XY)	z^2 in ***Z***, (XY)	*does LastZ*
11	**Z^X**	Hybrid Y^X	z^n = r^n * exp(i*n*Arg)	x in X reg; z in Y,Z regs	z^x in ***Z***, (XY)	*does LastZ*
12	**Z=0?**	Is z=0?	is z=0?	z in ***Z*** (XY)	YES/NO (skips if false)	
13	**Z=I?**	Is z=I?	is z=i?	z in ***Z*** (XY)	YES/NO (skips if false)	
14	**Z=W?**	Is z=w?	is z=w?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
15	**Z=WR?**	are z & w equal if rounded?	is Rnd(z)=Rnd(w)?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
16	**Z#0?**	is z equal to zero?	is z#0?	z in ***Z*** (XY)	YES/NO (skips if false)	
17	**Z#W?**	Is z equal to w?	is z=w?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
18	**ZACOS**	Complex ACOS	acos z = pi/2 - asin z	z in ***Z*** (XY)	acos(z) in ***Z*** (XY)	*does LastZ*
19	**ZALOG**	Complex 10^X	e^[z*ln(10)]	z in ***Z*** (XY)	10^z in ***Z*** (X,Y) and ALPHA	*does LastZ*
20	**ZASIN**	Complex ASIN	asin z = -i * asinh (iz)	z in ***Z*** (XY)	asin(z) in ***Z***, (XY)	*does LastZ*
21	**ZATAN**	Complex ATAN	atan z = -i * atanh (iz)	z in ***Z*** (XY)	atan(z) in ***Z*** (XY)	*does LastZ*
22	**ZCOS**	Complex COS	cos z = cosh (iz)	z in ***Z*** (XY)	cos(z) in ***Z*** (XY)	*does LastZ*
23	**ZDBL**	Calculates 2*Z (Doubles it)	2z = 2x + 2iy	z in ***Z*** (XY)	2*z in ***Z*** (XY)	*does LastZ*
24	**ZEXP**	Complex e^X	e^x * e^(iy)	z in ***Z*** (XY)	e^z in ***Z*** (XY) and ALPHA	*does LastZ*
25	**ZFRC**	Fractional Parts	FRC[Re(z)]; FRC[Im(z)]	z in ***Z*** (XY)	result in ***Z*** (XY)	*does LastZ*
26	**ZHACOS**	Complex Hyp. ACOS	acosh z = Ln[z + SQ(z^2 - 1)]	z in ***Z*** (XY)	acosh(z) in ***Z*** (XY)	*does LastZ*
27	**ZHALF**	Calculates Z/2 (halves it)	z/2 = (x/2 + iy/2)	z in ***Z*** (XY)	Halves z in ***Z*** (XY)	*does LastZ*
28	**ZHASIN**	Complex Hyp. ASIN	asinh z = Ln[z + SQ(z^2 + 1)]	z in ***Z*** (XY)	asinh(z) in ***Z*** (XY)	*does LastZ*
29	**ZHATAN**	Complex Hyp. ATAN	atanh z = 1/2 * Ln[(1+z)/(1-z)]	z in ***Z*** (XY)	atanh(z) in ***Z*** (XY)	*does LastZ*
30	**ZHCOS**	Complex Hyp. COS	cosh z = 1/2 * [e^z + e^-z]	z in ***Z*** (XY)	cosh(z) in ***Z*** (XY)	*does LastZ*
31	**ZHSIN**	Complex Hyp. SIN	sinh z = 1/2 * [e^z - e^-z]	z in ***Z*** (XY)	sinh(z) in ***Z*** (XY)	*does LastZ*
32	**ZHTAN**	Complex Hyp. TAN	tanh z = (e^z-e^-z)/(e^z+e^-z)	z in ***Z*** (XY)	tanh(z) in ***Z*** (XY)	*does LastZ*
33	**ZIMAG?**	is Im(z)=0?	is Im(z)=0?	z in ***Z*** (XY)	YES/NO (skips if false)	
34	**ZIN?**	Is z inside the unit circle?	is |z|<1?	z in ***Z*** (XY)	YES/NO (skips if false)	

35	**ZINT**	Integers Z	INT[(Re(z)] & INT[Im(z)]	z in **Z** (XY)	Integer Re and Im in **Z** (XY)	*does LastZ*
36	**ZINV**	Complex Inversion	x/(x^2 + y^2) - i y/(x^2 + y^2)	z in **Z** (XY)	1/z in **Z** (XY) and ALPHA	*does LastZ*
37	**ZLN**	Complex LN	ln(z) = ln(r) + i*Arg	z in **Z** (XY)	Ln(z) in **Z** (XY)	*does LastZ*
38	**ZLOG**	Complex LOG	log(z) = ln(z)/ln(10)	z in **Z** (XY)	Log(z) in **Z** (X,Y)	*does LastZ*
39	**ZNEG**	Complex CHS	-z = -x - iy	z in **Z** (XY)	-z in **Z** (XY)	*does LastZ*
40	**ZOUT?**	Is z outside the unit circle?	is |z|>1?	z in **Z** (XY)	YES/NO (skips if false)	
41	**ZREAL?**	Is Re(z)=0?	Is Re(z)=0?	z in **Z** (XY)	YES/NO (skips if false)	
42	**ZRND**	Rounds Z to display settings	rounded values to display	z in **Z** (XY)	Rounded Re & Im in **Z** (XY)	*does LastZ*
43	**ZSIN**	Complex SIN	sin z = -i *sinh (iz)	z in **Z** (XY)	sin(z) in **Z** (XY)	*does LastZ*
44	**ZSQRT**	Complex SQRT (Direct)	sqr(z)=sqr(r) * e^(i*Arg/2)	z in **Z** (XY)	main value of z^1/2 in **Z** (XY)	*does LastZ*
45	**ZTAN**	Complex TAN	tan z = - i * tanh (iz)	z in **Z** (XY)	tan(z) in **Z** (XY)	*does LastZ*
46	**ZUNIT?**	Is z on the unit circle?	is |z|=1?	z in **Z** (XY)	YES/NO (skips if false)	
47	**-ZSTACK**	***Section Header***	***n/a***	***none***	*Shows "Running..." msg*	
48	**CLZ**	Clears Z	Re(z)=0=Im(z)	none	**Z** level (XY) cleared	
49	**CLZST**	Clears Z-Stack	n/a	none	Z-Stack Cleared	
50	**LASTZ**	Complex LASTX	n/a	none	Last z in X,Y regs;	*Lifts Buffer*
51	**ZAVIEW**	Shows Complex Z	n/a	z in **Z** (XY)	Shows z in ALPHA	
52	**ZENTER^**	Copies Z into the W register	n/a	z in **Z** (XY)	Pushes z one level Up	*Lifts Buffer*
53	**Z<>** _ _	Complex Exchange	n/a	Reg# as suffix	Exchanges Z with regs contents	*Prompting*
54	**Z<>ST** _ _	Exchanges Z and L#	n/a	z in XY, level# in prompt	z in L#; L# in L1 & X,Y	*Prompting*
55	**ZTRP**	Exchanges Re(Z) and Im(Z)	zTrp = y + iX	z in **Z** (XY)	Im(z) in X, Re(z) in Y	*does LastZ*
56	**Z<>W**	Exchange Z and W (L2)	n/a	w in **W**, z in **Z** (XY)	z in L2 & Z,T w in L1 & X,Y	
57	**ZIMAG^**	Enter imaginary number	n/a	Im(z) in X	zero in X; Im(z) in Y	*Lifts Buffer*
58	**ZRCL** _ _	Complex RCL	n/a	Reg# as suffix	z in X,Y - lifts stack	*Lifts Buffer, Prompting*
59	**ZRDN**	Z-Stack Roll Down	n/a	Stack Levels	Rolls Down stack	*Drops Buffer*
60	**ZREAL^**	Enter Real number in Z	n/a	Re(z) in X	Re(z) in X;, Zero in Y	*Lifts Buffer*
61	**ZRPL^**	Replicates z in all levels	L4=L3=L2=L1	z in **Z** (XY)	z in all 4 levels	*Lifts Buffer*
62	**ZRUP**	Z-Stack Roll Up	n/a	Stack Levels	Rolls Up stack	*Lifts Buffer*
63	**ZSTO** _ _	Complex STO	n/a	Reg# as suffix	Stores z in consequtive regs	*Prompting*
64	**ZVIEW** _ _	Complex View	n/a	Reg# as suffix	Shows z in ALPHA	*Prompting*
1	**^IM/AG** _	***Natural Data Entry***	***Re ^ IM or r ^ arg***	***Prompts for Im(z)***	*z in **Z** (XY), stack lifted*	*Prompting, Lifts Buffer*
2	**1/Z**	alternative ZINV (Uses TOPOL)	1/r * exp(-i arg)	z in **Z** (XY)	1/z in X,Y registers and ALPHA	*does LastZ*
3	**e^Z**	alternative ZEXP	e^z = e^x * (cos y + i sin y)	z in **Z** (XY)	exp(z) in **Z** (XY)	*does LastZ*
4	**NXTACS**	Next ACOS Value	z1,2 = +/- z0 + 2π	z0 in **Z** (XY)	z1 in **W**, z2 in **Z** (XY)	*does LastZ*
5	**NXTASN**	Next ASIN Value	z1,2 = +/- z0 + 2π/2	z0 in **Z** (XY)	z1 in **W**, z2 in **Z** (XY)	*does LastZ*
6	**NXTATN**	Next ATAN value	z1,2 = z0 +/- π	z0 in **Z** (XY)	z1 in **W**, z2 in **Z** (XY)	*does LastZ*
7	**NXTLN**	Next Ln(z)	next(k) = Ln(z) + 2kπ J	LN(z) in **Z** (XY) regs	z1 in **W**, z2 in **Z** (XY)	*does LastZ*

<table>
<tr><td>8</td><td>NXTRTN</td><td>Next Complex Root</td><td>next(k) = z^1/n * e^(2kπ/n J)</td><td>n in X reg.; z^1/n in Z,Y regs</td><td>z^1/n * e^(2π/n J) in Z (XY)</td><td>does LastZ</td></tr>
<tr><td>9</td><td>SQRTZ</td><td>Alternative SQRT (Uses TOPOL)</td><td>sqr(z)=sqr(r) * e^(i*Arg/2)</td><td>z in Z (XY)</td><td>main value of z^1/2 in Z (XY)</td><td>does LastZ</td></tr>
<tr><td>10</td><td>X^1/Z</td><td>Hybrid Y^X</td><td>a^1/z = exp(1/ z*Ln a)</td><td>x in X reg; z in Y,Z regs</td><td>x^z in Z (XY)</td><td>does LastZ</td></tr>
<tr><td>11</td><td>Z^3</td><td>Cubic power</td><td>z=z^3</td><td>z in Z (Im in Y, Re in X)</td><td>result in Z (XY)</td><td>more accurate than Z^X</td></tr>
<tr><td>12</td><td>ZCHSX</td><td>Sign Change by X</td><td>(-1)^n * z</td><td>x in X reg; z in Y,Z regs</td><td>{(-1)^x * z} in Z (XY)</td><td>does LastZ</td></tr>
<tr><td>13</td><td>ZIMAG</td><td>Make z Imaginary</td><td>Re(z)=0</td><td>z in Z (Im in Y, Re in X)</td><td>zero in X; Im(z) in Y</td><td>does LastZ</td></tr>
<tr><td>14</td><td>ZINT?</td><td>Checks if Z is an integer number</td><td>are Im(z)=0 and FRC[Re(z)]=0?</td><td>z in Z (Im in Y, Re in X)</td><td>YES/NO (skips if false)</td><td>used in Bessel fncs</td></tr>
<tr><td>15</td><td>ZK?YN</td><td>Block Key Assignments</td><td>n/a</td><td>prompt-driven</td><td>Makes / Removes assignments</td><td>may do PACKING</td></tr>
<tr><td>16</td><td>ZKBRD _</td><td>Complex keyboard launcher</td><td>n/a</td><td>Prompt-driven</td><td>Launches function</td><td>prompting, launcher</td></tr>
<tr><td>17</td><td>ZMTV</td><td>Multi-Value Complex functions</td><td>see manual</td><td>z in Z (XY)</td><td>Menu-driven</td><td>FOCAL</td></tr>
<tr><td>18</td><td>ZREAL</td><td>Make Z Real</td><td>Im(z)=0</td><td>z in Z (XY)</td><td>Re(z) in X;, Zero in Y</td><td>does LastZ</td></tr>
<tr><td>19</td><td>ZST+ _ _</td><td>STO Addition</td><td>cR + z</td><td>z in Z (XY)</td><td>Adds z to complex register#</td><td>prompting</td></tr>
<tr><td>20</td><td>ZST- _ _</td><td>STO Subtraction</td><td>cR - z</td><td>z in Z (XY)</td><td>Subtract z from complex register#</td><td>prompting</td></tr>
<tr><td>21</td><td>ZST* _ _</td><td>STO Multiply</td><td>cR * z</td><td>z in Z (XY)</td><td>Multiplies z to complex register#</td><td>prompting</td></tr>
<tr><td>22</td><td>ZST/ _ _</td><td>STO Divide</td><td>cR / z</td><td>z in Z (XY)</td><td>Divides complex register by z</td><td>prompting</td></tr>
<tr><td>23</td><td>ZWLOG</td><td>Base-w Logarithm</td><td>base w in W, arg. In Z</td><td>w in W, z in Z (XY)</td><td></td><td>Drops Buffer, LastZ</td></tr>
<tr><td>24</td><td>-ZVECTOR</td><td>Section Header</td><td>n/a</td><td>none</td><td>Displays Revision Number</td><td></td></tr>
<tr><td>25</td><td>HARM</td><td>Harmonic numbers</td><td>Σ(1/k) , k=1,2…n</td><td>n in X</td><td>H(n) in X, x in LastX</td><td>used in Bessel fncs</td></tr>
<tr><td>26</td><td>POLAR</td><td>Sets POLAR mode on</td><td>sets the Polar flag in Buffer</td><td>none</td><td>shows Re(z)+J Im(z)</td><td></td></tr>
<tr><td>27</td><td>RECT</td><td>Sets RECT mode on</td><td>clears the Polar flag in Buffer</td><td>none</td><td>shows r <) arg</td><td></td></tr>
<tr><td>28</td><td>Z*I</td><td>Multiplies by I (90 deg. Rotation)</td><td>iz = -Im(z) + I Re(z)</td><td>z in Z (XY)</td><td>z*i in L1 & XY</td><td>does LastZ</td></tr>
<tr><td>29</td><td>ZARG</td><td>Argument of Z</td><td>atan(y/x)</td><td>z in Z (XY)</td><td>Arg(z) in X, (Y reg void)</td><td>zeroes Y, LastZ</td></tr>
<tr><td>30</td><td>ZCONJ</td><td>Complex Conjugate</td><td>conj = x - iy</td><td>z in Z (XY)</td><td>Inverts sign of Im(z)</td><td>does LastZ</td></tr>
<tr><td>31</td><td>ZMOD</td><td>Module of Z</td><td>|z|=sqr(x^2+y^2)</td><td>z in Z (XY)</td><td>Mod(z) in X, (Y reg void)</td><td>zeroes Y, LastZ</td></tr>
<tr><td>32</td><td>ZNORM</td><td>Norm of Z (I.e. square of Module)</td><td>||z||=|z| ^2</td><td>z in Z (XY)</td><td>(mod(z)^2) in X,Y</td><td>zeroes Y, LastZ</td></tr>
<tr><td>33</td><td>ZPOL</td><td>Converts to Polar notation</td><td>R-P</td><td>z in Z (XY)</td><td>Mod(z) in X; Arg(z) in Y</td><td>does LastZ</td></tr>
<tr><td>34</td><td>ZREC</td><td>Convers to Rectangular notation</td><td>P-R</td><td>Mod(z) in X; Arg(z) in Y</td><td>Re(z) in X; Im(z) in Y</td><td>does LastZ</td></tr>
<tr><td>35</td><td>ZSIGN</td><td>Complex SIGN</td><td>sign = z/|z|</td><td>z in Z (XY)</td><td>z/Mod(z) in X,Y</td><td>does LastZ</td></tr>
<tr><td>36</td><td>ZWANG</td><td>Angle between Z and W</td><td>arg(zw) = Arg(z) - Arg(w)</td><td>z in Z (XY)</td><td>ang(z,w) in X (Y void)</td><td>Drops Buffer LastZ</td></tr>
<tr><td>37</td><td>ZWCROSS</td><td>Cross product of Z and W</td><td>z x w = |z| *|w| *Sin(Angle)</td><td>w in W, z in Z (XY)</td><td>z x w in X (Y void)</td><td>Drops Buffer LastZ</td></tr>
<tr><td>38</td><td>ZWDET</td><td>Determinant of Z and W</td><td>|zw| = x2*y1 - y2*x1</td><td>w in W, z in Z (XY)</td><td>det(z,w) in X (Y void)</td><td>Drops Buffer LastZ</td></tr>
<tr><td>39</td><td>ZWDIST</td><td>Distance between Z and W</td><td>|w-z| = SQR[(x2-x1)^2 - (y2-y1)^2]</td><td>w in W, z in Z (XY)</td><td>dist(z,x) in X (Y void)</td><td>Drops Buffer LastZ</td></tr>
<tr><td>40</td><td>ZWDOT</td><td>Dot product of Z and W</td><td>z*w = x1*x2 + y1*y2</td><td>w in W, z in Z (XY)</td><td>dot(z,w) in X, (Y void)</td><td>Drops Buffer LastZ</td></tr>
<tr><td>41</td><td>ZWLINE</td><td>Line equation defined by Z and W</td><td>a=(y1-y2) / (x1-x2)</td><td>w in W, z in Z (XY)</td><td>y=ax+b in ALPHA; b in Y, a in X</td><td>Drops Buffer LastZ</td></tr>
<tr><td>42</td><td>-HL ZMATH</td><td>Section Header</td><td>Calculates 2^x-1</td><td>x in X</td><td>Result in X</td><td>used in ZZETA</td></tr>
<tr><td>43</td><td>EIZ/IZ</td><td>spherical hankel h1(0,z)</td><td>h^(1)(0,z) = exp(i*z) / i*z</td><td>z in Z (XY)</td><td>result in Z (XY)</td><td>does LastZ</td></tr>
<tr><td>44</td><td>GEUZ</td><td>Euler's gamma constant</td><td>γ=0,577215665</td><td>none</td><td>γ constant as complex</td><td>Lifts Buffer</td></tr>
</table>

45	**ZAWL**	Inverse of Lambert W	z* e^z	z in ***Z*** (XY)	result in ***Z*** (XY)	*does LastZ*
46	**ZBS**	Bessel subroutine 1st. Kind	see manual	w in W, z/2 in Z	iterative SUM	FOCAL
47	**ZBS2**	Bessel subroutine 2nd. Kind	see manual, Flag 6 controls case	w in cR00, z/2 in cR01	iterative SUM	FOCAL
48	**ZCRT**	Complex Cubic Eq. Roots	Cubic ecuation roots	A,B,C,D in Z-Stack	roots in ***V***, ***W***, and ***Z*** (XY) levels	FOCAL
49	**ZGAMMA**	Complex Γ(z) for z#0, -1, -2…	Lanczos approximation	z in ***Z*** (XY)	Γ(z) in ***Z*** (XY)	*uses reflection for Re(z)<0*
50	**ZGPRD**	Partial calculation of Gamma	PROD(z+k); k=0,1…6	z in ***Z*** (XY)	result in ***Z*** (XY)	*does LastZ*
51	***"ZIBS"***	Bessel I function	see manual	w in ***W***, z in ***Z*** (XY)	I(w,z) in ***Z*** (XY)	FOCAL
52	***"ZJBS"***	Bessel J function	see manual	w in ***W***, z in ***Z*** (XY)	J(w,z) in ***Z*** (XY)	FOCAL
53	***"ZKBS"***	Bessel K function	see manual	w in ***W***, z in ***Z*** (XY)	K(w,z) in ***Z*** (XY)	FOCAL
54	**ZLNG**	Gamma Logarithm function	Stirling method w/ correction	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
55	**ZPIX**	Product by pi	z*π	z in ***Z*** (XY)	result in ***Z*** (XY)	*more accurate FOCAL*
56	**ZPROOT**	Roots of complex polynomials	Iterative	Prompt-driven	roots in ***W*** and ***Z*** (XY) levels	FOCAL
57	**ZPSI**	Complex Digamma	Approximation	z in ***Z*** (XY)	Psi(z) in X,Y regs. And ALPHA	FOCAL
58	**ZQRT**	Complex Quadratic Eq. Roots	Quadratic ecuation roots	A,B,C in Zstack	Calculates roots of equation	FOCAL
59	***"ZSHK1"***	Spherical Hankel h1	$h^{(1)}(w,z)$	order w in W; arg. z in Z	result in ***Z*** (XY)	FOCAL
60	***"ZSHK2"***	Spherical Hankel h2	$h^{(2)}(w,z)$	order w in W; arg. z in Z	result in ***Z*** (XY)	FOCAL
61	**ZSOLVE**	Solves for F(z)=0	Newton's method	Fnc. name in R06	Calculates one root for f(z)	FOCAL
62	***"ZWL"***	Lambert W function	see manual	z in ***Z*** (XY)	W(z) in ***Z*** (XY)	FOCAL
63	***"ZYBS"***	Bessel Y function	see manual	w in ***W***, z in ***Z*** (XY)	Y(w,z) in ***Z*** (XY)	FOCAL
64	**ZZETA**	Riemann Zeta function	Borwein Algorithm	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
1	**-ZBUFFER**	***Section Header***	n/a	None	None	
2	**CLZB**	Clears Z buffer	n/a	None	buffler cleared	
3	**L1=XY?**	is L1 equal to XY?	n/a	None	Y/N, skip if false	
4	**L1<>L _ _**	Swap L1 & Level	*n/a*	Level# as suffix	levels exchanged	*Prompting*
5	**L1<>L2**	Swap L1 & L2	n/a	None	levels exchanged	
6	**L1<>L3**	Swap L1 & L3	n/a	None	levels exchanged	
7	**L1<>L4**	Swap L1 & L4	n/a	None	levels exchanged	
8	**L1<>LX**	Swap L1 & Level	n/a	level in X	levels exchanged	
9	**L2=ZT?**	is L2 equal to ZT?	n/a	None	Y/N, skip if false	
10	**L2>ZT**	Copies L2 into ZT	n/a	None	L2 copied to ZT	
11	**LVIEW _**	View Level	n/a	Level# as suffix	*Transposed value!*	*Prompting*
12	**LVIEWX**	View level by X	n/a	level in X	*Transposed value!*	
13	**PREMON**	Copies XY into L0 and finds Zbuffer	n/a	Re(z) in X; Im(z) in Y	none	
14	**PSTMON**	Copies XY into L1 and synch's up	n/a	Re(z) in X; Im(z) in Y	None	
15	**RG>ZB _ _**	Copies registers to Z buffer	n/a	Reg# as suffix	data copied from registers	*Prompting*
16	**ST>ZB**	Copies real stack to L1 & L2	n/a	None	stack copied to buffer	

17	**XY>L _**	Copies XY into Level	n/a	Level# as suffix	XY copied to LEVEL	*Prompting*
18	**XY>L0**	Copies XY into L0	n/a	Re(z) in X; Im(z) in Y	XY copied to L0	
19	**XY>L1**	Copies XY into L1	n/a	Re(z) in X; Im(z) in Y	XY copied to L1	
20	**ZB>RG _ _**	copies buffer to registers	n/a	Reg# as suffix	data copied to registers	*Prompting*
21	**ZB>ST**	Copies L1 & L2 into real stack	n/a	None	buffer copied to Stack	
22	**ZBDROP**	Drops Z buffer one level	n/a	None	levels dropped	Drops Buffer
23	**ZBHEAD**	Zbuffer Header info	n/a	None	header register in ALPHA	
24	**ZBLIFT**	Lifts Z buffer one level	n/a	None	buffer lifted	*Lifts Buffer*
25	***ZBVIEW***	Shows Z Buffer	n/a	None	shows header & all levels	FOCAL
26	**-B UTILS**	***Section Header***	n/a	None	None	
27	**B?**	Does buffer exist?	n/a	buffer id# in X	YES/NO (skips if false)	CCD Module
28	**BLIST**	lists all buffers existing	n/a	none	list in Alpha	D. Yerka
29	**BLNG?**	Buffer length	n/a	buffer id# in X	buffer size in X	CCD Module
30	**BX>RG**	copies buffer to registers	n/a	buffer id# in X	data copied into R00 to end	David Assm
31	**CLB**	Clear buffer	n/a	buffer id# in X	Clears buffer from memory	CCD Module
32	**FINDBX**	finds buffer address	n/a	buffer id# in X	buffer address in X	D. Yerka
33	**MAKEBX**	makes buffer in RAM	n/a	(id#,size) in X	buffer created	D. Yerka
34	**RG>BX**	copies registers to buffer	n/a	Data in R00 to Rnn	Copied to Buffer	David Assm

(*) Buffer functions have been moved to the BUFFERLAND Module, under a dedicated section for the 41Z case.

Appendix 5.- Buffer logic function table.

#	Function	Description	Pre-Exec: Alpha in XY	Pre-Exec: XY to L0	Pre-Exec: XY to L1	Pre-Exec: Buffer LIFT	Pre-Exec: L2 -> ZT		Post-Exec: Buffer DROP	Post-Exec: XY into L1	Post-Exec: L1,2 -> XYZT	Post-Exec: ZAVIEW	
1	- HP-41 Z	Initialize Buffer	yes	***no***	***yes***	no	no		no	no	no	yes	
2	W^Z	Power	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	*yes*	*POSTDUAL*
3	Z+	Addition	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	*yes*	*POSTDUAL*
4	Z-	Substraction	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	*yes*	*POSTDUAL*
5	Z*	Multiply	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	*yes*	*POSTDUAL*
6	Z/	Divide	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	*yes*	*POSTDUAL*
7	ZWANG	Angle between	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
8	ZWCROSS	Cross Product	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
9	ZWDET	Determinat	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
10	ZWDIST	Distance	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
11	ZWDOT	Dot Product	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
12	ZWLINE	Line Equation	yes	yes	no	no	***yes***	*PREDUAL*	***yes***	*yes*	*yes*	***no***	***PSTDUAL-2***
13	Z=W?	is Z=W?	yes	***no***	***yes***	no	***yes***	***PREDUL-2***	no	no	no	no	
14	Z=WR?	is Z=W round?	yes	***no***	***yes***	no	***yes***	***PREDUL-2***	no	no	no	no	
15	Z#W?	is Z not W?	yes	***no***	***yes***	no	***yes***	***PREDUL-2***	no	no	no	no	
16	Z=0?	is Z Zero?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
17	Z#0?	is Z not zero?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
18	Z=I?	is Z = i?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
19	ZREAL?	Is Z real?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
20	ZIMAG?	Is Z imag?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
21	ZIN?	|Z|<1?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
22	ZOUT?	|Z|>1?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
23	ZUNIT?	|Z|=1?	yes	***no***	***yes***	no	*no*	***PREMON-2***	no	no	no	no	
24	X^Z	Hybrid Power	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
25	Z^2	Z^2	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
26	Z^X	Z^X	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
27	ZACOS	ACOS	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
28	ZACOSH	ACOSH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
29	ZALOG	10^Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
30	ZASIN	ASIN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON

31	**ZASINH**	ASINH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
32	**ZATAN**	ATAN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
33	**ZATANH**	ATANH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
34	**ZCONJ**	X-Yj	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
35	**ZCOS**	COS	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
36	**ZCOSH**	COSH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
37	**ZDBL**	2*Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
38	**ZEXP**	E^Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
39	**ZHALF**	Z/2	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
40	**ZINV**	1/Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
41	**ZLN**	Ln(Z)	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
42	**ZINT**		yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
43	**ZFRC**		yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
44	**ZLOG**	Log(Z)	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
45	**ZNEG**	-Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
46	**ZRND**	rounded Z	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
47	**ZSIGN**	Sign(Z)	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
48	**ZSIN**	SIN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
49	**ZSINH**	SINH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
50	**ZSQRT**	Square Root	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
51	**ZTAN**	TAN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
52	**ZTANH**	TANH	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
53	**ZTRP**	Re<>Im	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
54	**ZARG**	Zarg	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	***no***	***PSTMON-2***
55	**ZMOD**	**|Z|**	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	***no***	***PSTMON-2***
56	**ZNORM**	|Z|^2	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	***no***	***PSTMON-2***
57	**ZREC**	Rectangular	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
58	**ZPOL**	Polar Notation	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
59	**e^Z**	alternate ZEXP	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
60	**EIZ/IZ**	function	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
61	**Z^1/X**	hybrid power	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
62	**Z*I**	rotation	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
63	**Z/I**	rotation	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
64	**NXTASN**	Next ASIN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
65	**NXTACS**	Next ACOS	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON
66	**NXTATN**	Next ATAN	yes	yes	no	no	no	PREMON	no	*yes*	*yes*	*yes*	POSTMON

67	NXTLOG	Next LN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
68	NXTNRT	Next Nth. Root	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
69	ZAVIEW	Output Z	yes	no	no	no	no		no	no	no	yes	
70	CLZ	Clear Z	no	no	no	no	no		no	yes	yes	yes	POSTMON
71	ZIMAG	Clear Re(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
72	ZREAL	Clear Im(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
73	CLZST	Clear Zstack	no	no	no	no	no		no	no	yes	yes	PSTMON-3
74	Z<>	Exchange	yes	no	no	no	no	PREMON	no	yes	yes	yes	POSTMON
75	Z<>W	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
76	Z<>R	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
77	Z<>S	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
78	LASTZ	last argument	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
79	ZR^	Roll Up Zstack	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
80	ZRCL	Recall to Z	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
81	IMAGINE	inputs Im(z)	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
82	ZENTER^	Enter level	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
83	ZREAL^	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
84	ZIMAG^	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
85	ZRDN	Roll Down ZSTK	yes	no	yes	no	no	PREMON-2	yes	no	yes	yes	PSTMON-3
86	ZREPL	Replicates Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
87	ZSTO	Stores Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3